MLG 2006

Proceedings of the
International Workshop on
Mining and Learning with Graphs

in conjunction with ECML/PKDD 2006

Thomas Gärtner
Gemma C. Garriga
Thorsten Meinl
(Eds.)

Berlin, Germany, 18th September 2006

Preface

At a time where the amount of data collected day by day far exceeds the human capabilities to extract the knowledge hidden in it, it becomes more and more important to automate the process of learning. Typical data collections have two things in common: They are huge and the information stored in them is highly structured.

Graphs are one of the most popular data representations in mathematics, computer science, engineering disciplines, and other natural sciences. This workshop on "Mining and Learning with Graphs" (MLG) thus concentrated on learning from graphs and its subclasses such as — but not limited to — trees, sequences (GTS). The primary goal of MLG was to bring together researchers working on various aspects of mining and learning with graphs. It is hence in the tradition of previous ECML/PKDD workshops on "Mining Graphs, Trees, and Sequences" (MGTS) but extends its scope to include other areas of machine learning and data mining also concerned with graphs and their subclasses such as:

- Algorithmic aspects of
- Theoretical aspects of
- Open problems in
- Novel applications of
- Evaluative studies of

the following — non-exclusive — list of topics

- Kernels and Distances for graphs.
- GTS-structured output spaces.
- Frequent GTS mining.
- Learning with generative GTS models and compact (e.g., intensional) representations like GTS transformations, grammars, or matchings.
- Theoretical aspects of learning from GTS.
- Probabilistic modelling of GTS.
- GTS-based approaches to transductive and semi-supervised learning.

Compared to previous workshops on MGTS, MLG was able to attract a record number of submissions (28 full and 6 short papers). For time and space restrictions, we could only accept 9 full papers and 15 short papers (most of which were originally submitted as full papers).

For the first time, the workshop received the support of the PASCAL Network of Excellence that sponsored the invited talk and the best paper award. We most sincerely thank the PASCAL network for this sponsoring; the program committee and additional reviewers for their reviews; as well as the authors for their high quality submissions.

Thomas Gärtner, Gemma C. Garriga, Thorsten Meinl

Workshop Co-Chairs

Thomas Gärtner
Fraunhofer IAIS, Sankt Augustin, Germany
thomas.gaertner@ais.fraunhofer.de

Gemma C. Garriga
Univeristat Politècnica de Catalunya, Barcelona, Spain
garriga@lsi.upc.edu

Thorsten Meinl
University of Konstanz, Germany
Thorsten.Meinl@uni-konstanz.de

Program Commitee

Yasemin Altun, Toyota Technological Institute at Chicago, USA

José Balcázar, Universitat Politècnica de Catalunya, Spain

Hendrik Blockeel, Katholieke Universiteit Leuven, Belgium

Christian Borgelt, University of Magdeburg, Germany

Horst Bunke, University of Bern, Switzerland

Tiberio Caetano, National ICT, Australia

Ingrid Fischer, University of Konstanz, Germany

Peter Flach, University of Bristol, UK

Paolo Frasconi, Università degli Studi di Firenze, Italy

Thore Graepel, Microsoft Research Cambridge, UK

Thomas Hofmann, TU Darmstadt, Germany

Thorsten Joachims, Cornell University, USA

Roni Khardon, Tufts University, USA

Kristian Kersting, University of Freiburg, Germany

Stefan Kramer, Technical University Munich, Germany

Jure Leskovec, Carnegie Mellon University, USA

Brian Milch, University of California in Berkeley, USA

Siegfried Nijssen, University of Freiburg, Germany

Alex J. Smola, National ICT, Australia

György Turán, University of Illinois at Chicago, USA

Additional Reviewers

Table of Contents

Full Papers

Short Papers

Intersection Algorithms and a Closure Operator on Unordered Trees

José L. Balcázar, Albert Bifet and Antoni Lozano

Universitat Politècnica de Catalunya,
{balqui,abifet,antoni}@lsi.upc.edu

Abstract. Link-based data may be studied formally by means of unordered trees. On a dataset formed by such link-based data, a natural notion of support-based closure can be immediately defined. Abstracting information from subsets of such data requires, first, a formal notion of intersection; second, deeper understanding of the notion of closure; and, third, efficient algorithms for computing intersections on unordered trees. We provide answers to these three questions.

1 Introduction

Closure-based mining is well-established by now as one of the various approaches to summarize subsets of a large dataset. Sharing some of the attractive features of frequency-based summarization of subsets, it offers an alternative view with both downsides and advantages; among the latter, there are the facts that, first, by imposing closure, the number of frequent sets is heavily reduced and, second, the possibility appears of developing a mathematical foundation that connects closure-based mining with lattice-theoretic approaches like Formal Concept Analysis.

Closure-based mining on itemsets is, by now, well understood, and there are interesting algorithmic developments; thus, there have been subsequent efforts in moving towards closure-based mining on structured data, particularly sequences, trees and graphs; see the survey [4] and the references there. One of the differences with closed itemset mining stems from the fact that the set theoretic intersection no longer applies, and whereas the intersection of sets is a set, the intersection of two sequences or two trees is not one sequence or one tree. This makes it nontrivial to justify the word "closed" in terms of a standard closure operator. Many papers resort to a support-based notion of closedness of a tree or sequence ([5], see below); others (like [1]) choose a variant of trees where a closure operator between trees can be actually defined (via least general generalization). In some cases, the trees are labeled, and strong conditions are imposed on the label patterns (such as nonrepeated labels in tree siblings [10] or nonrepeated labels at all in sequences [8]).

Here we attempt at formalizing a closure operator for substantiating the work on closed trees, with no resort to the labelings: we focus on the case where the given dataset consists of unordered, unlabeled, rooted trees; thus, our only

relevant information is the root and the link structure (so that the appropriate notion of subtree, so-called top-down subtree, preserves root and links), and solving the intersection problem along the same lines as in [3]. Thus, we only focus on the basic operations needed to phrase closure-based mining on such structures, but with a mathematically very demanding approach. We first formalize our structures and the notion of a tree being contained in another. We also evaluate the quantity of such combinatorial structures. We then move on to start our study of closure-based mining. Following the same path as in [7], we first need a notion of intersection: we will see that a natural notion of intersection of trees gives rise to intersection sets of trees, rather than individual trees, as with the sequences in [7]. We study the cardinality of such intersection sets, which we prove can be exponential in the worst case, although preliminary experiments suggest that intersection sets of cardinality beyond 1 hardly ever arise unless looked for.

We then propose a notion of Galois connection with the associated closure operator, in such a way that we can characterize support-based notions of closure with a mathematical operator. We complete this paper with preliminary algorithmic studies. We propose a natural recursive algorithm to compute intersections, and a more sophisticated method following a dynamic programming scheme; preliminary comparisons suggest that the dynamic programming algorithm is several orders of magnitude faster.

2 Preliminaries

2.1 Definitions

We will deal with rooted undirected trees with nodes of unbounded arity. This kind of trees will be referred throughout the paper simply as *trees*. The set of all trees will be denoted with $\mathcal{T}$. Additionally, we call *binary tree* a tree whose nodes have a maximum of two children. The letter $\mathcal{D} \subset \mathcal{T}$ will represent a finite list of data trees, sometimes treated as a set.

A tree t' is a *subtree* of a tree t (written $t' \preceq t$) if t' is a connected subgraph of t which contains the root of t (this is also known as *top-down subtree*). We say that $t_1, \ldots, t_k$ are the *components* of tree t if t is made of a node (the root) joined to the roots of all the t_i's. The components form a set, not a sequence; therefore, permuting them does not give a different tree. In our drawings, we follow the convention that larger trees are drawn at the left of smaller trees. In the algorithms we will identify trees by strings in the following way, which will allow us to order trees lexicographically (the drawing convention corresponds to using the lexicographically least identification).

Definition 1. *We define the injective total function* $\langle\cdot\rangle : \mathcal{T} \to \{0,1\}^*$ *recursively as follows. If* t *is a single node, then* $\langle t\rangle = 01$. *Otherwise, suppose that* $t_1, \ldots, t_k$ *are the components of* t *enumerated so that* $\langle t_1\rangle \leq \langle t_2\rangle \leq \ldots \leq \langle t_k\rangle$ *(in lexicographical order). Then* $\langle t\rangle = 0\langle t_1\rangle \ldots \langle t_k\rangle 1$. *We also define* $[\cdot] : \{0,1\}^* \to \mathcal{T}$ *such that for any tree* t, $[\langle t\rangle] = t$, *and is undefined for strings not in the image of* $\langle\cdot\rangle$.

The one-node tree [01] will be represented with the symbol •, and the two-node tree [0011] by •–•.

Definition 2. *Given two trees, a common subtree is a tree that is subtree of both; it is a maximal common subtree if it is not a subtree of any other common subtree; it is a maximum common subtree if there is no common subtree of larger size.*

Two trees have always some maximal common subtree but, as is shown in Figure 1, this common subtree does not need to be unique.

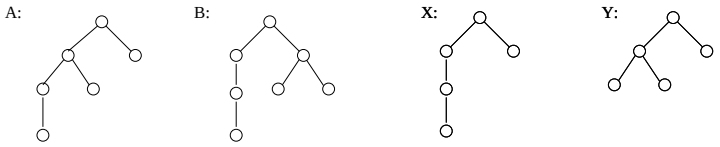


Fig. 1. Trees X and Y are maximal common subtrees of A and B.

In fact, trees X and Y have the maximum number of nodes among the common subtrees of A and B. As is shown in Figure 2, just a slight modification of A and B gives two maximal common subtrees of different sizes, showing that the concepts of maximal and maximum common subtree do not coincide in general.

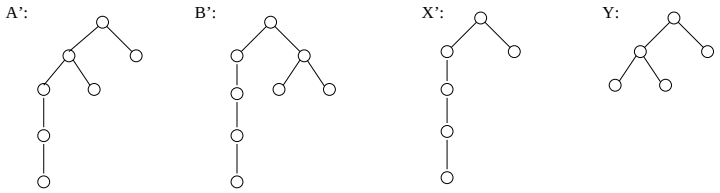


Fig. 2. Both X' and Y are maximal common subtrees of A' and B', but only X' is maximum.

2.2 Number of trees

The number of trees with n nodes is known to be $\Theta(\rho^n n^{-3/2})$, where $\rho = 0.3383218569$ ([9]). We provide a more modest lower bound based on an easy way to count the number of binary trees; this will be enough to show in a few lines an exponential lower bound on the number of trees with n nodes.

Define B_n as the number of binary trees with n nodes, and set $B_0 = 1$ for convenience. Clearly, a root without children is the only binary tree with one

node, so $B_1 = 1$. Now, B_n is the sum of all products $B_i B_j$ for every way to express $n-1$ as $i+j$ (meaning that the $n-1$ nodes other than the root are distributed into two subtrees having i and j nodes). So, we have

$$B_n = \sum_{\substack{i+j=n-1 \\ i \le j}} B_i B_j = \sum_{i=0}^{\lfloor \frac{n-1}{2} \rfloor} B_i B_{n-i-1}.$$

The second summation can be rewritten as

$$B_n = B_0 B_{n-1} + \sum_{i=0}^{\lfloor \frac{n-1}{2} \rfloor - 1} B_{i+1} B_{n-i-2} = B_{n-1} + \sum_{i=0}^{\lfloor \frac{n-3}{2} \rfloor} B_{i+1} B_{(n-2)-(i+1)-1}$$

which implies that $B_n \ge B_{n-1} + B_{n-2}$, thus showing that B_n is bigger than the n-th Fibonacci number F_n (note that the initial values also satisfy the inequality, since $F_0 = 0$ and $F_1 = F_2 = 1$). Since it is well-known that $F_{n+2} \ge \phi^n$, where $\phi > 1.618$ is the golden number, we have the lower bound

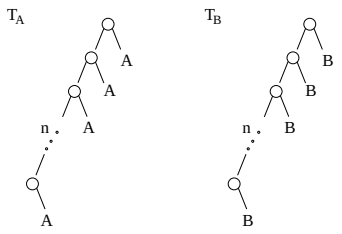
$$\phi^{n-2} \le F_n \le B_n.$$

which is also a lower bound for the total number of trees with n nodes.

2.3 Number of subtrees

We can easily observe, using the trees A, B, X, and Y of Section 2.1, that two trees can have an exponential number of maximal common subtrees.

Recall that the aforementioned trees have the property that X and Y are two maximal common subtrees of A and B. Now, consider the pair of trees constructed in the following way using copies of A and B. First, take a path of length $n-1$ (thus having n nodes which include the root and the unique leaf) and "attach" to each node a whole copy of A. Call this tree T_A. Then, do the same with a fresh path of the same length, with copies of B hanging from their nodes, and call this tree T_B. Graphically:



All the trees constructed similarly with copies of X or Y attached to each node of the main path (instead of A or B) are maximal common subtrees of T_A

and T_B. The fact that the copies are at different depths assures that all the 2^n possibilities correspond to different subtrees. Therefore, the number of different maximal common subtrees of T_A and T_B is at least 2^n (which is exponential in the input since the sum of the sizes of T_A and T_B is $15n$). Any algorithm for computing maximal common subtrees has, therefore, a worst case exponential cost due to the size of the output.

3 Closure operator and mining closed trees

Once a proper notion of intersection is available, we move on to build a notion of closed sets of trees, with a view towards a data mining framework operating on tree-structured data.

For a notion of closed (sets of) trees to make sense, we expect to be given as data a finite set (actually, a list) of transactions, each of which consisting of its transaction identifier (tid) and an unordered tree. Transaction identifiers are assumed to run sequentially from 1 to N, the size of the dataset. We denote $\mathcal{D} \subset \mathcal{T}$ the dataset.

The support of a tree t in $\mathcal{D}$ is the number of transactions where t is a subtree of the tree in the transaction. General usage would lead to the following notion of closed tree:

Definition 3. *A tree t is closed for $\mathcal{D}$ if no tree $t' \neq t$ exists with the same support such that $t \preceq t'$.*

Note that $t \preceq t'$ implies that t is a subtree of all the transactions where t' is a subtree, so that the support of t is, at least, that of t'. Existence of a larger t' with the same support would mean that t does not gather all the possible information about the transactions in which it appears, since t' also appears in the same transactions and gives more information (is more specific). A closed tree is maximally specific for the transactions in which it appears. However, note that the example of the trees A and B given above provides two trees X and Y with the same support, and yet mutually incomparable.

We aim at clarifying the properties of closed trees, providing a more detailed justification of the term "closed" through a closure operator obtained from a Galois connection, along the lines of [6], [3], [7], or [2] for unstructured or otherwise structured datasets. However, given that the intersection of a set of trees is not a single tree but yet another set of trees, we will find that the notion of "closed" is to be applied to subsets of the transaction list, and that the notion of a "closed tree" t is not exactly coincident with the singleton $\{t\}$ being closed.

3.1 Galois Connection

A Galois connection is provided by two functions, relating two lattices in a certain way. Here our lattices are plain power sets of the transactions, on the one hand, and of the corresponding subtrees, in the other. On the basis of the binary relation $t \preceq t'$, the following definition and proposition are rather standard.

Definition 4. *The Galois connection pair:*

- *For finite* $A \subseteq \mathcal{D}$, $\sigma(A) = \{t \in \mathcal{T} \mid \forall\, t' \in A\,(t \preceq t')\}$
- *For finite* $B \subset \mathcal{T}$, *not necessarily in* $\mathcal{D}$, $\tau_{\mathcal{D}}(B) = \{t' \in \mathcal{D} \mid \forall\, t \in B\,(t \preceq t')\}$

There are many ways to argue that such a pair is a Galois connection. One of the most useful ones is as follows.

Proposition 1. *For all finite* $A \subseteq \mathcal{D}$ *and* $B \subset \mathcal{T}$, *the following holds:*

$$A \subseteq \tau_{\mathcal{D}}(B) \iff B \subseteq \sigma(A)$$

Proof. By definition, each of the two sides is equivalent to

$$\forall\, t \in B\, \forall\, t' \in A\,(t \preceq t') \qquad \square$$

It is well-known that the compositions (in either order) of the two functions that define a Galois connection constitute a closure operator, that is, are monotonic, extensive, and idempotent (with respect, in our case, to set inclusion).

Corollary 1. $\Gamma_{\mathcal{D}} = \tau_{\mathcal{D}} \circ \sigma$ *is a closure operator on the subsets of* $\mathcal{D}$.

Thus, we have now a concept of closed sets of trees; however, the notion of closure based on support as previously defined corresponds to single trees, and it is worth clarifying the connection between them, naturally considering the closure of the singleton set containing a given tree, $\Gamma_{\mathcal{D}}(\{t\})$. We point out the following easy-to-check properties:

1. $t \in \Gamma_{\mathcal{D}}(\{t\})$
2. $t' \in \Gamma_{\mathcal{D}}(\{t\})$ if and only if $\forall s \in \mathcal{D}(t \preceq s \Rightarrow t' \preceq s)$
3. t is maximal in $\Gamma_{\mathcal{D}}(\{t\})$ (that is, $\forall t' \in \Gamma_{\mathcal{D}}(\{t\})[t \preceq t' \Rightarrow t = t']$) if and only if $\forall t'(\forall s \in \mathcal{D}[t \preceq s \Rightarrow t' \preceq s] \wedge t \preceq t' \Rightarrow t = t')$

The definition of closed tree can be phrased in a similar manner as follows: t is closed for $\mathcal{D}$ if and only if: $\forall t'(t \preceq t' \wedge \mathrm{supp}(t) = \mathrm{supp}(t') \Rightarrow t = t')$.

Theorem 1. *A tree* t *is closed for* $\mathcal{D}$ *if and only if it is maximal in* $\Gamma_{\mathcal{D}}(\{t\})$.

Proof. Suppose t is maximal in $\Gamma_{\mathcal{D}}(\{t\})$, and let $t \preceq t'$ with $\mathrm{supp}(t) = \mathrm{supp}(t')$. The data trees s that count for the support of t' must count as well for the support of t, because $t' \preceq s$ implies $t \preceq t' \preceq s$. The equality of the supports then implies that they are the same set, that is, $\forall s \in \mathcal{D}(t \preceq s \iff t' \preceq s)$, and then, by the third property above, maximality implies $t = t'$. Thus t is closed.

Conversely, suppose t closed and let $t' \in \Gamma_{\mathcal{D}}(\{t\})$ with $t \preceq t'$. Again, then $\mathrm{supp}(t') \leq \mathrm{supp}(t)$; but, from $t' \in \Gamma_{\mathcal{D}}(\{t\})$ we have, as in the second property above, $(t \preceq s \Rightarrow t' \preceq s)$ for all $s \in \mathcal{D}$, that is, $\mathrm{supp}(t) \leq \mathrm{supp}(t')$. Hence, equality holds, and from the fact that t is closed, with $t \preceq t'$ and $\mathrm{supp}(t) = \mathrm{supp}(t')$, we infer $t = t'$. Thus, t is maximal in $\Gamma_{\mathcal{D}}(\{t\})$. $\square$

Now we can continue the argument as follows. Suppose t is maximal in some closed set of trees B. From $t \in B$, by monotonicity and idempotency, together with aforementioned properties, we obtain $t \in \Gamma_{\mathcal{D}}(\{t\}) \subseteq \Gamma_{\mathcal{D}}(B) = B$; being maximal in the larger set implies being maximal in the smaller one, so that t is maximal in $\Gamma_{\mathcal{D}}(\{t\})$ as well. Hence, we have argued the following alternative, somewhat simpler, characterization:

Theorem 2. *A tree is closed for $\mathcal{D}$ if and only if it is maximal in some closed set of $\Gamma_{\mathcal{D}}$.*

Yet another simpler observation is that each closed set is uniquely defined through its maximal elements. In fact, our implementations chose to avoid duplicate calculations and redundant information by just storing the maximal trees of each closed set. We could have defined the Galois connection so that it would provide us "irredundant" sets of trees by keeping only maximal ones; the property of maximality would be then simplified into $t \in \Gamma_{\mathcal{D}}(\{t\})$, which would not be guaranteed anymore (cf. the notion of stable sequences in [3]). The formal details of the validation of the Galois connection property would differ slightly (in particular, the ordering would not be simply a mere subset relationship) but the essentials would be identical, so that we refrain from developing that approach here; we would obtain a development somewhat closer to [3] than our current development is. But there would be no indisputable advantages.

4 Intersection algorithms

Computing a potentially large intersection of a set of trees is not a trivial task, given that there is no ordering among the components: a maximal element of the intersection may arise through mapping smaller components of one of the trees into larger ones of the other. Therefore, the degree of branching along the exploration is high.

4.1 Finding the intersection recursively

We start with a straightforward algorithm for finding all maximal common subtrees of two trees in a recursive way. The basic idea is to exploit the recursive structure of the problem by considering all the ways to match the components of the two input trees. Suppose we are given the trees t and r, whose components are $t_1, \ldots, t_k$ and $r_1, \ldots, r_n$, respectively. If $k \leq n$, then clearly $(t_1, r_1), \ldots, (t_k, r_k)$ is one of those matchings. Then, we recursively compute the maximal common subtrees of each pair (t_i, r_i) and "cross" them with the subtrees of the previously computed pairs, thus giving a set of maximal common subtrees of t and r for this particular identity matching. The algorithm explores all the (exponentially many) matchings and, finally, eliminates repetitions and trees which are not maximal (by using recursion again).

We do not specify the data structure used to represent the trees. The only condition needed is that every component t' of a tree t can be accessed with

```
RECURSIVE INTERSECTION(r, t)
 1  if (r = •) or (t = •)
 2     then S ← {•}
 3  elseif (r = •-•) or (t = •-•)
 4     then S ← {•-•}
 5     else  S ← {}
 6           n_r ← #COMPONENTS(r)
 7           n_t ← #COMPONENTS(t)
 8           for each m in MATCHINGS(n_r, n_t)
 9                do mTrees ← {•}
10                   for each (i, j) in m
11                        do c_r ← COMPONENT(r, i)
12                           c_t ← COMPONENT(t, j)
13                           cTrees ← RECURSIVE INTERSECTION(c_r, c_t)
14                           mTrees ← CROSS(mTrees, cTrees)
15                   S ← MAX SUBTREES(S, mTrees)
16  return S
```

Fig. 3. Algorithm RECURSIVE INTERSECTION

an index which indicates the lexicographical position of its encoding $\langle t' \rangle$ with respect to the encodings of the other components; this will be COMPONENT(t, i). The other procedures are as follows:

- #COMPONENTS(t) computes the number of components of t, this is, the arity of the root of t.
- MATCHINGS(n_1, n_2) computes the set of perfect matchings of the graph K_{n_1,n_2}, this is, of the complete bipartite graph with partition classes $\{1, \ldots, n_1\}$ and $\{1, \ldots, n_2\}$ (each class represents the components of one of the trees). For example,
  MATCHINGS$(2, 3)$ $= \{\{(1,1),(2,2)\}, \{(1,1),(2,3)\}, \{(1,2),(2,1)\}, \{(1,2),(2,3)\}, \{(1,3),(2,1)\}, \{(1,3),(2,2)\}\}$.
- CROSS(l_1, l_2) returns a list of trees constructed in the following way: for each tree t_1 in l_1 and for each tree t_2 in l_2 make a copy of t_1 and add t_2 to it as a new component.
- MAX SUBTREES(S_1, S_2) returns the list of trees containing every tree in S_1 and every tree in S_2 that is not a subtree of another tree in S_1, thus leaving only the maximal subtrees. There is a further analysis of this procedure in the next subsection.

The fact that, as has been shown, two trees may have an exponential number of maximal common subtrees necessarily makes any algorithm for computing all maximal subtrees inefficient. However, there is still space for some improvement.

4.2 Finding the intersection by dynamic programming

In the above algorithm, recursion can be replaced by a table of precomputed answers for the components of the input trees. This way we avoid repeated recursive calls for the same trees, and speed up the computation. Suppose we are given two trees r and t. In the first place, we compute all the trees that can appear in the recursive queries of Recursive Intersection(r, t). This is done in the following procedure:

- Subcomponents(t) returns a list containing t if $t = \bullet$; otherwise, if t has the components $t_1, \ldots, t_k$, then, it returns a list containing t and the trees in Subcomponents(t_i) for every t_i, ordered increasingly by number of nodes.

The new algorithm shown in Figure 4 constructs a dictionary D accessed by pairs of trees (t_1, t_2) (or, more precisely, by their codes $(\langle t_1 \rangle, \langle t_2 \rangle)$), when the input trees are nontrival (different from $\bullet$ and $\bullet\!\!-\!\!\bullet$). Inside the main loops, the trees which are used as keys for accessing the dictionary are taken from the lists Subcomponents(r) and Subcomponents(t), where r and t are the input trees.

```
Dynamic Programming Intersection(r, t)
 1  for each s_r in Subcomponents(r)
 2       do for each s_t in Subcomponents(t)
 3            do if (s_r = •) or (s_t = •)
 4                 then D[s_r, s_t] ← {•}
 5               elseif (s_r = •–•) or (s_t = •–•)
 6                 then D[s_r, s_t] ← {•–•}
 7                 else  D[s_r, s_t] ← {}
 8                       ns_r ← #Components(s_r)
 9                       ns_t ← #Components(s_t)
10                       for each m in Matchings(ns_r, ns_t)
11                            do mTrees ← {•}
12                               for each (i, j) in m
13                                    do cs_r ← Component(s_r, i)
14                                       cs_t ← Component(s_t, j)
15                                       cTrees ← D[cs_r, cs_t]
16                                       mTrees ← Cross(mTrees, cTrees)
17                               D[s_r, s_t] ← Max Subtrees(D[s_r, s_t], mTrees)
18  return D[r, t]
```

Fig. 4. Algorithm Dynamic Programming Intersection

Note that the fact that the number of trees in Subcomponents(t) is linear in the number of nodes of t assures a quadratic size for D. The entries of the dictionary are computed by increasing order of the number of nodes; this way, the

```
MAX SUBTREES(S_1, S_2)
1  for each r in S_1
2        do for each t in S_2
3           if r is a subtree of t
4              then mark r
5           elseif t is a subtree of r
6              then mark t
7  return sublist of nonmarked trees in S_1 ∪ S_2
```

Fig. 5. Algorithm MAX SUBTREES

information needed to compute an entry has already been computed in previous steps.

The procedure MAX SUBTREES, which appears in the penultimate step of the two intersection algorithms presented, is shown in Figure 5. The key point in the procedure MAX SUBTREES is the identification of subtrees made in steps 3 and 5. By standard algorithms, it can be decided whether $t_1 \preceq t_2$ in time $O(n_1 n_2^{1.5})$ ([11]), where n_1 and n_2 are the number of nodes of t_1 and t_2, respectively.

Finally, the table in Figure 6 shows an example of the intersections stored in the dictionary by the algorithm DYNAMIC PROGRAMMING INTERSECTION with trees A and B of Figure 1 as input.

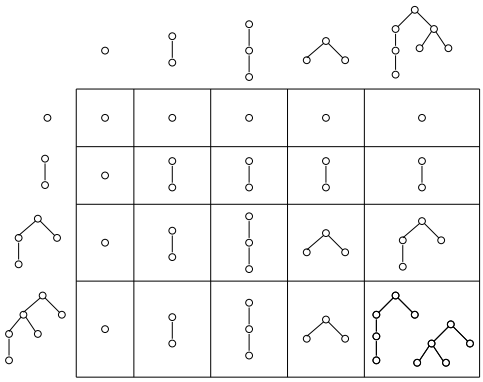


Fig. 6. Table with all partial results computed

5 Conclusion

Closure-based structures have been proposed in several references in a context of data mining. They may allow for summarizing the (huge) lattice of all the

subsets of a dataset by reducing it to only closure sets: these may add up to a much lesser quantity, and each closed subset of the dataset may offer some sort of actionable interpretation. We have studied such an approach to tree-like link structures.

Whereas we do not attempt at the design of specific algorithms here for computing closures yet, we have pointed out that the notion of closure given in the previous section does provide the appropriate framework for a closure-based data mining task on tree-structured data. Moreover, the properties established here suggest that it is possible to construct the lattice of closed sets of trees by mining first the closed trees and, then, organizing them into the desired lattice.

We describe a toy example of the closure lattice for a simple dataset consisting of six trees, thus providing additional hints on our notion of intersection; these were not made up for the example, but were instead obtained through six different (rather arbitrary) random seeds of the synthetic tree mining generator of Zaki [12].

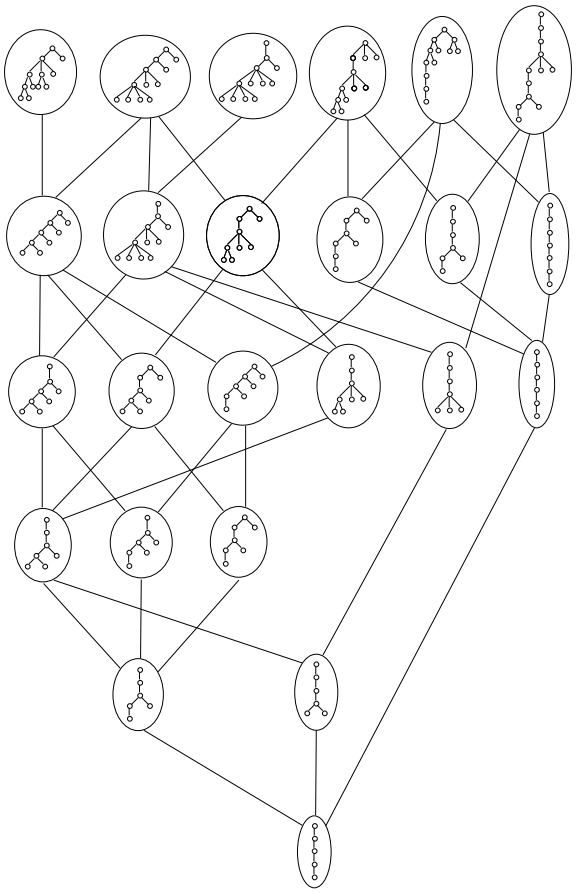


Fig. 7. Lattice of closed trees for the six input trees in the top row

The figure depicts the closed sets obtained. It is interesting to note that all the intersections came up to a single tree, a fact that suggests that the exponential blow-up of the intersections sets, which is possible as explained in a previous section, appears infrequently enough. Of course, the common intersection of the whole dataset is (at least) a "pole" whose length is the minimal height of the data trees.

The study of algorithmics for the construction of this lattice, or of a fragment thereof (e.g. through frequency thresholds) since it will be usually quite large, will be subject of further work, as well as the corresponding notions of implications or association rules for the framework of unordered trees.

References

1. Hiroki Arimura and Takeaki Uno. An output-polyunomial time algorithm for mining frequent closed attribute trees. In *ILP*, pages 1–19, 2005.
2. Jaume Baixeries and José L. Balcázar. Discrete deterministic data mining as knowledge compilation. In *Workshop on Discrete Math. and Data Mining at SIAM DM Conference*, 2003.
3. José L. Balcázar and Gemma C. Garriga. On Horn axiomatizations for sequential data. In *ICDT*, pages 215–229 (extended version to appear in Theoretical Computer Science), 2005.
4. Yun Chi, Richard Muntz, Siegfried Nijssen, and Joost Kok. Frequent subtree mining – an overview. *Fundamenta Informaticae*, XXI:1001–1038, 2001.
5. Yun Chi, Yi Xia, Yirong Yang, and Richard Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *IEEE Trans. Knowl. Data Eng.*, 17(2):190–202, 2005.
6. B. Ganter and R. Wille. *Formal Concept Analysis*. Springer-Verlag, 1999.
7. Gemma C. Garriga. Formal methods for mining structured objects. PhD Thesis, 2006.
8. Gemma C. Garriga and José L. Balcázar. Coproduct transformations on lattices of closed partial orders. In *ICGT*, pages 336–352, 2004.
9. J. M. Plotkin and John W. Rosenthal. How to obtain an asymptotic expansion of a sequence from an analytic identity satisfied by its generating function. *J. Austral. Math. Soc. (Series A)*, 56:131–143, 1994.
10. Alexandre Termier, Marie-Christine Rousset, and Michele Sebag. DRYADE: a new approach for discovering closed frequent trees in heterogeneous tree databases. In *ICDM*, pages 543–546, 2004.
11. Gabriel Valiente. *Algorithms on Trees and Graphs*. Springer-Verlag, Berlin, 2002.
12. Mohammed J. Zaki. Efficiently mining frequent trees in a forest. In *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

Discriminative Identification of Duplicates

Peter Haider, Ulf Brefeld, and Tobias Scheffer

Humboldt Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
{haider,brefeld,scheffer}@informatik.hu-berlin.de

Abstract. The problem of finding duplicates in data is ubiquitous in data mining. We cast the problem of finding duplicates in sequential data into a poly-cut problem on a fully connected graph. The edge weights can be identified with parameterized pairwise similarities between objects that are optimized by structural support vector machines on labeled training sets. Our approach adapts the similarity measure to the data and is independent of the number of clusters. We present three large margin approximations of learning the pairwise similarities: an integrated QP-formulation, a sequential multi-class approach and a pairwise classifier. We report on experimental results.

1 Introduction

The problem of identifying duplicates has applications ranging from recognizing objects from different perspectives and angles to the identification of objects that are intentionally altered to obfuscate their true identity, origin, or purpose. This occurs, for instance, in the context of email spam and virus detection.

Spam and virus senders avoid mailing identical copies of their messages because it would be an easy giveaway. Identifying a batch of messages would allow email service providers to hold back the entire batch, and to identify hijacked servers that are being used to disseminate spam or viruses. Therefore, spam senders generate messages according to templates. Table 1 shows an example of two spam messages that have been generated with a spamming tool. Slots of a common template are filled according to a grammar; the tool also applies obfuscation techniques such as random insertions of spaces.

In the database community, the "database deduping problem" is another popular instance of the duplicate identification problem. Other occurrences of the problem include named entity resolution, and the grouping of images that show, for instance, the same person.

A natural approach to identifying duplicates is to group similar objects together by a cluster algorithm. However, prominent algorithms like k-means or Expectation Maximization require the number of clusters beforehand. Moreover, given a problem at hand, it is often ambiguous to decide whether two objects are similar or not.

Correlation clustering [3] meets our requirements by accounting for potentially infinitely many clusters. Its solution is equivalent to a maximum poly-cut

Hello, This is Terry Hagan.We are accepting your mo rtgage application. Our company confirms you are legible for a $250.000 loan for a $380.00/month. Approval process will take 1 minute, so please fill out the form on our website: http://www.competentagent.com/application/ Best Regards, Terry Hagan; Senior Account Director Trades/Fin ance Department North Office
Dear Mr/Mrs, This is Brenda Dunn.We are accepting your mortga ge application. Our office confirms you can get a $228.000 lo an for a $371.00 per month payment. Follow the link to our website and submit your contact information. Easy as 1,2,3. http://www.competentagent.com/application/ Best Regards, Brenda Dunn; Accounts Manager Trades/Fin ance Department East Office

Table 1. Two spam mails from the same batch.

in a fully connected graph spanned by the objects and their pairwise similarities [11].

We address the problem of learning a duplicate detection hypothesis from labeled data. That is, we start from data in which all elements that are duplicates of one another have been tagged as such. This allows us to learn the similarity function that parameterizes the clustering model such that it correctly groups the duplicates in the training data. The similarity measure can be learned by structural SVMs in a discriminative way.

We firstly derive a loss augmented optimization problem that can be solved directly. Due to a cubic number of variables, solving this initial problem is hardly tractable for large data sets. Secondly, we present an approach that makes use of the sequential nature of the objects and thirdly, we approximate the optimal solution by a pairwise classifier. Experiments detail characteristics of all three methods.

The rest of our paper is structured as follows. We report on related work in Section 2 and introduce our problem setting together with the decoding strategy in Section 3. We present support vector algorithms for identifying duplicates in Section 4 and report on experimental results in Section 5. Section 6 concludes.

2 Related Work

The identification of duplicates has been studied with fixed similarity measures, such as the fraction of matching words [9, 8] and sentences [6]. Other applications

include the identification of duplicates in data bases [5], and in centralized [14] and decentralized networks [23].

Correlation clustering on fully connected graphs is introduced in [2, 3]. A generalization to arbitrary graphs is presented in [7] and [11] shows the equivalence to a poly-cut problem. Approximation strategies to the NP-complete decoding are presented in [10, 17]. Finley and Joachims [13] investigated supervised clustering with structural support vector machines.

Prior information about the cluster structure of a data set allows for enhancements to classical clustering algorithms such as k-means. E.g., Wagstaff et al. [21] incorporate the background knowledge as must-link and cannot-link constraints into the clustering process, while [4, 22] learn a metric over the data space that incorporates the prior knowledge.

Several discriminative algorithms have been studied that utilize joint spaces of input and output variables; these include max-margin Markov models [18], kernel conditional random fields [15], hidden Markov support vector machines [1], and support vector machines for structured output spaces [20]. These methods utilize kernels to compute the inner product in input output space. This approach allows to capture arbitrary dependencies between inputs and outputs. An application-specific learning method is constructed by defining appropriate features, and choosing a decoding procedure that efficiently calculates the argmax, exploiting the dependency structure of the features.

3 Preliminaries

The task is to find a model f such that given a set of instances $\mathbf{x}$ the true partitioning $\mathbf{y}$ given as an adjacency matrix yields the highest score

$$\mathbf{y} = \operatorname*{argmax}_{\bar{\mathbf{y}} \in \mathcal{Y}} f(\mathbf{x}, \bar{\mathbf{y}}). \tag{1}$$

We measure the quality of f by an appropriate, symmetric, nonnegative loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_0^+$ that details the distance between the true partition $\mathbf{y}$ and the prediction $\hat{\mathbf{y}} = \operatorname{argmax}_{\bar{\mathbf{y}}} f(\mathbf{x}, \bar{\mathbf{y}})$. A natural measure for two clusterings is the Rand index [16]. The corresponding loss function Δ_{Rand} is given by

$$\begin{aligned} \Delta_{Rand}(\mathbf{y}, \hat{\mathbf{y}}) &= 1 - Q_{Rand}(\mathbf{y}, \hat{\mathbf{y}}) \\ &= 1 - \frac{\sum_{j,k<j} [[y_{jk} = \hat{y}_{jk}]]}{|\mathbf{y}|} \\ &= \sum_{j,k<j} \frac{[[y_{jk} \neq \hat{y}_{jk}]]}{|\mathbf{y}|}, \end{aligned}$$

where $[[\sigma]]$ is the indicator function which yields 1 if the proposition σ is true and 0 otherwise. We can restate the optimization problem as finding a function f that minimizes the expected risk

$$R(f) = \int_{\mathcal{X} \times \mathcal{Y}} \Delta_{Rand}(\mathbf{y}, \operatorname{argmax}_{\bar{\mathbf{y}}} f(\mathbf{x}, \bar{\mathbf{y}})) dP(\mathbf{x}, \mathbf{y}) \tag{2}$$

where $P(\mathbf{x}, \mathbf{y})$ is the (unknown) distribution of sets of objects and their clusterings. As in the classical setting we address this problem by searching a minimizer of the empirical risk given by

$$R_S(f) = \frac{1}{n} \sum_{i=1}^{n} \Delta_{Rand}(\mathbf{y}^{(i)}, \operatorname{argmax}_{\bar{\mathbf{y}}} f(\mathbf{x}, \bar{\mathbf{y}})), \tag{3}$$

regularized by $\|f\|^2$.

Correlation clustering [3] maintains a symmetric similarity matrix whose elements denote pairwise similarities between objects. This representation allows to recast the problem as a poly-cut problem in a fully connected graph, where objects are identified with nodes and edges are weighted with the respective pairwise similarities. The optimal partitioning can either be found by minimizing the edge weights between clusters of objects or by maximizing the edge weights within clusters of objects. Following the latter leads to the integer optimization problem

$$\hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}} \; f(\mathbf{x}, \mathbf{y}) \;=\; \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}} \sum_{j=1}^{|\mathbf{x}|} \sum_{k=1}^{j-1} y_{jk} sim(x_j, x_k) \tag{4}$$

where y_{jk} indicates wether x_j and x_k belong to the same cluster. The set $\mathcal{Y}$ contains all equivalence relations over $\mathbf{x}$ given as an adjacency matrix, that is, all $\mathbf{y}$ which satisfy the triangle inequality $(1 - y_{jk}) + (1 - y_{kl}) \geq (1 - y_{jl})$ where $y_{jk} \in \{0, 1\}$. The maximum is attained by the partitioning $\mathbf{y}$ that maximizes the within-cluster similarities. We follow [13] and use a parameterized similarity measure between two objects x_j and x_k given by

$$sim(x_j, x_k) = \sum_{t=1}^{T} w_t \phi_t(x_j, x_k) = \mathbf{w}^\top \Phi(x_j, x_k), \tag{5}$$

where $\Phi(x_j, x_k) = (..., \phi_t(x_j, x_k), ...)^\top$ is the similarity vector of x_j and x_k; e.g., in our running example $\phi_{234}(x_j, x_k)$ might be an indicator function that equals 1 if both mails are of the same mime-type. Substituting 5 into 4 shows that we can rewrite f as a generalized linear model in joint input output space

$$\begin{aligned} f(\mathbf{x}, \mathbf{y}) &= \sum_{j=1}^{|\mathbf{x}|} \sum_{k=1}^{j-1} y_{jk} sim(x_j, x_k) && (6) \\ &= \sum_{j=1}^{|\mathbf{x}|} \sum_{k=1}^{j-1} y_{jk} \mathbf{w}^\top \Phi(x_j, x_k) && (7) \\ &= \mathbf{w}^\top \underbrace{\left(\sum_{j=1}^{|\mathbf{x}|} \sum_{k=1}^{j-1} y_{jk} \Phi(x_j, x_k) \right)}_{=: \Psi(\mathbf{x}, \mathbf{y})} && (8) \\ &= \mathbf{w}^\top \Psi(\mathbf{x}, \mathbf{y}). && (9) \end{aligned}$$

In the following we will refer to a sample S of n input output pairs $(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$, drawn *i.i.d.* according to $P(\mathbf{x}, \mathbf{y})$. The i-th pair contains $|\mathbf{x}^{(i)}| = m_i$ instances $x_1^{(i)}, \ldots, x_{m_i}^{(i)}$ with adjacency matrix $\mathbf{y}^{(i)}$ such that $y_{jk}^{(i)} = 1$ if $x_j^{(i)}$ and $x_k^{(i)}$ are in the same partition. We denote the set of all adjacency matrices of possible partitionings of the i-th set by $\mathcal{Y}^{(i)}$.

4 Discriminative Identification of Duplicates

In this section we present three discriminative approaches to the identification of duplicates: an integrated QP-formulation, a sequential multi-class approach, and a pairwise classifier.

4.1 Integrated Optimization Problem

Bansal et al. [3] show that exact inference is NP-complete. However, the optimal solution can be approximated by substituting real valued edge weights $z_{jk} \in [0, 1]$ for the integer valued edge weights $y_{jk} \in \{0, 1\}$. The decoding problem in Equation 4 can be solved approximately by the following decoding strategy.

Decoding Strategy 1 *Given m instances $x_1, \ldots, x_m \in \mathcal{X}$ and a similarity measure $sim_{\mathbf{w}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Over all values $\mathbf{z} \in \mathbb{R}^m$ maximize $\sum_{j=1}^{m} \sum_{k=1}^{j-1} z_{jk} sim(x_j, x_k)$ subject to the constraints $\forall_{j,k,l} \, (1 - z_{jk}) + (1 - z_{kl}) \geq (1 - z_{jl})$ and $\forall_{j,k} \, 0 \leq z_{jk} \leq 1$.*

The substitution of the approximate labels, gives rise to the loss function $\Delta_{Rand}(\mathbf{y}, \mathbf{z}) = \sum_{j,k<j} (|y_{jk} - z_{jk}|)/|\mathbf{y}|$. The optimization problem of the structural support vector machine in terms of approximate labels $\mathbf{z}$ can be stated as follows.

Optimization Problem 1 *Given n labeled clusterings, loss function Δ_{Rand}, $C > 0$; over all $\mathbf{w}$ and ξ_i minimize $||\mathbf{w}||^2 + C \sum_{i=1}^{n} \xi_i$ subject to the constraints $\forall_{i=1}^{n} \mathbf{w}^\top \Psi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) + \xi_i \geq \max_{\mathbf{z} \in \mathcal{Z}^{(i)}} \left[\mathbf{w}^\top \Psi(\mathbf{x}^{(i)}, \mathbf{z}) + \Delta_{Rand}(\mathbf{y}^{(i)}, \mathbf{z}) \right]$ and $\forall_{i=1}^{n} \xi_i \geq 0$, where $\mathcal{Z}^{(i)}$ consists of all possible approximate labelings of $\mathbf{x}^{(i)}$ which satisfy the triangle inequality.*

Similar to [19] the loss can be integrated into the decoding of the top scoring clustering. This gives us

$$\begin{aligned} \max_{\mathbf{z}_i} \quad & d^{(i)} + \sum_{j,k<j} \mathbf{z}_{i,jk} (\mathbf{w}^\top \Phi(x_j^{(i)}, x_k^{(i)}) - e_{jk}^{(i)}) \\ \text{s.t.} \quad & \forall_{j,k,l} \quad (1 - z_{i,jk}) + (1 - z_{i,kl}) \geq (1 - z_{i,jl}), \\ & \forall j, k \quad 0 \leq z_{i,jk} \leq 1, \end{aligned}$$

where $d^{(i)} = \frac{\sum_{j,k<j} y_{jk}^{(i)}}{|y^{(i)}|}$ and $e_{jk}^{(i)} = \frac{2 y_{jk}^{(i)} - 1}{|y^{(i)}|}$. Integrating the constraint into the objective function leads to the corresponding Lagrangian

$$L(\mathbf{z}_i, \lambda_i, \nu_i, \kappa_i) = d^{(i)} + \nu_i^\top \mathbf{1} + \lambda_i^\top \mathbf{1} + \left[\mathbf{w}^\top \Phi(\mathbf{x}^{(i)}) - \mathbf{e}^{(i)} - A^{(i)} \lambda_i^\top - \nu_i + \kappa_i \right]^\top \mathbf{z}_i$$

where the coefficient matrix $A^{(i)}$ is defined as

$$A^{(i)}_{jkl,j'k'} := \begin{cases} +1 & : \text{ if } (j' = j \wedge k' = k) \vee (j' = k \wedge k' = l) \\ -1 & : \text{ if } j' = j \wedge k' = l \\ 0 & : \text{ otherwise} \end{cases}$$

The substitution of the derivatives with respect to $\mathbf{z}_i$ into the Lagrangian and elimination of κ_i removes its dependence on the primal variables and we resolve the corresponding dual that is given by

$$\begin{aligned} \min_{\lambda_i, \nu_i} \quad & d^{(i)} + \nu_i^\top \mathbf{1} + \lambda_i^\top \mathbf{1} \\ \text{s.t. } & \mathbf{w}^\top \Phi(\mathbf{x}^{(i)}) - \mathbf{e}^{(i)} - A^{(i)} \lambda_i - \nu_i \leq \mathbf{0} \\ & \lambda_i, \nu_i \geq \mathbf{0}. \end{aligned}$$

Strong duality holds and the minimization over λ and ν can be combined with the minimization over $\mathbf{w}$. The reintegration into optimization problem 1 leads to the integrated Optimization Problem 2 that can be solved directly.

Optimization Problem 2 *Given n labeled clusterings, $C > 0$; over all $\mathbf{w}$, ξ_i, λ_i, and ν_i, minimize $||\mathbf{w}||^2 + C \sum_{i=1}^n \xi_i$ subject to the constraints 10-12.*

$$\forall_{i=1}^n \ \mathbf{w}^\top \Psi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) + \xi_i \geq d^{(i)} + \nu_i^\top \mathbf{1} + \lambda_i^\top \mathbf{1}, \tag{10}$$

$$\forall_{i=1}^n \ \mathbf{w}^\top \Phi(\mathbf{x}^{(i)}) - \mathbf{e}^{(i)} \leq A^{(i)} \lambda_i + \nu_i, \tag{11}$$

$$\forall_{i=1}^n \ \lambda_i, \nu_i \geq \mathbf{0}, \tag{12}$$

The number of Lagrange multipliers λ_i in Optimization Problem 2 is cubic in the number of instances m_i; i.e., its solution becomes intractable for large data sets. In the following two sections we present two approaches that overcome this drawback.

4.2 Sequential Clustering

Our second approach accounts for the sequential nature of the data. In the server-sided batch detection scenario incoming mails have to be classified immediately upon arrival. In our running example each incoming email is either grouped to an existing batch or it becomes its own singelton batch.

Therefore, it suffices to maintain a window that contains the last m incoming mails. As soon as a new mail arrives it is substituted for the oldest mail in the window and a new clustering is computed. The latter step can be approximated by finding a cluster or opening a new batch only for the latest mail, respectively. Algorithm 1 details this approach.

The adjacency matrix $\mathbf{y}$ can be obtained from the clustering $\mathcal{C}$ by $y_{jk}(\mathcal{C}) = [[\exists c \in \mathcal{C} : x_j \in c \wedge x_k \in c]]$. Given a fixed clustering of $x_1, \ldots, x_{m-1}$, the decoding

Algorithm 1 Sequential Clustering

```
1   C ← {}
2   for j = 1 ... |x|
3           c_j = argmax_{c∈C} Σ_{x_k∈c} w^⊤ Φ(x_k, x_j)
4           if Σ_{x_k∈c_j} w^⊤ Φ(x_k, x_j) < 0
5                   C ← C ∪ {{x_j}}
6           else
7                   C ← C \ {c_j} ∪ {c_j ∪ {x_j}}
8           endif
9   endfor
10  return C
```

problem in 4 reduces to

$$\max_{\mathbf{y}\in\mathcal{Y}} \sum_{j=1}^{m} \sum_{k=1}^{j-1} y_{jk} sim_{\mathbf{w}}(x_j, x_k) = \tag{13}$$

$$\max_{\mathbf{y}\in\mathcal{Y}} \sum_{j=1}^{m-1} \sum_{k=1}^{j-1} y_{jk} sim_{\mathbf{w}}(x_j, x_k) + \sum_{k=1}^{m-1} y_{mk} sim_{\mathbf{w}}(x_m, x_k). \tag{14}$$

The first summand of Equation 14 is constant; thus finding a cluster for x_m reduces to the Decoding Strategy 2, where the additional cluster $\bar{c}$ accounts for x_m being dissimilar to its predecessors in the window.

Decoding Strategy 2 *Given m instances $x_1, \ldots, x_m \in \mathcal{X}$, similarity measure $sim_{\mathbf{w}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, and a clustering $\mathcal{C}$ of instances $x_1, \ldots, x_{m-1}$; over all values $c \in \{\mathcal{C} \bigcup \bar{c}\}$ maximize $\sum_{x_k \in c} sim_{\mathbf{w}}(x_m, x_k)$.*

If we denote the set of all possible clusterings in which x_j is reassigned to any cluster by $\mathcal{C}_j$ we derive the following minimization problem.

Optimization Problem 3 *Given n labeled clusterings, $C > 0$; over all $\mathbf{w}$ and ξ_{ij}, minimize $\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i,j} \xi_{ij}$ subject to the constraints $\forall_{i=1}^{N}, \forall_{j=1}^{m_i}, \forall \hat{\mathcal{C}} \in \mathcal{C}_j^{(i)} \mathbf{w}^\top \Psi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) + \xi_{ij} \geq [\mathbf{w}^\top \Psi(\mathbf{x}^{(i)}, \mathbf{y}(\hat{\mathcal{C}})) + \Delta(\mathbf{y}^{(i)}, \mathbf{y}(\hat{\mathcal{C}}))]$*

Since the number of clusters is upper bounded by the window size, $|\mathcal{C}| \leq m$, Optimization Problem 3 has at most $n \cdot \sum_{i=1}^{n} m_i^2$ constraints and can be solved by standard techniques. This approach is equivalent to single-vector multi-class classification [12]. Also note that the obtained solution for the weight vector $\mathbf{w}$ is independent of the used decoding strategy, and can thus be used with every other approximation of correlation clustering as well.

4.3 Pairwise Classification

The multi-class approach can be further approximated by a binary classifier that outputs class $+1$ if two instances are similar and class -1 otherwise. Therefore,

we use all pairs of instances $(x_j^{(i)}, x_k^{(i)})$ within the training tuple $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ as inputs and define the labels $\upsilon_{jk}^{(i)} = +1$ if $y_{jk}^{(i)} = 1$, and $\upsilon_{jk}^{(i)} = -1$ if $y_{jk}^{(i)} = 0$. This leads us to the standard formulation of a binary support vector machine in Optimization Problem 4.

Optimization Problem 4 *Given n labeled clusterings, $C > 0$; over all $\mathbf{w}$ and ξ_{ijk}, minimize $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i,j,k}\xi_{ijk}$ subject to the constraints $\forall_{i=1}^{N}, \forall_{j=1}^{m_i}, \forall_{k=1}^{j-1} \upsilon_{jk}^{(i)}(\mathbf{w}^\top \Phi(x_j^{(i)}, x_k^{(i)}) + b) \geq 1 - \xi_{ijk}$.*

The weight vector $\mathbf{w}$ can directly be used as parameter of the similarity measure, i.e. the decision function of the binary classifier is equivalent to the pairwise similarity function. Analogously to the sequential clustering, the pairwise classification allows the use of any decoding strategy.

However, this approach suffers several drawbacks compared to the two previously devised solutions. Firstly, an application-specific loss function cannot be incorporated into the learning problem that implicitly minimizes the 0/1 error. Secondly, transitive dependencies within the training tuples are ignored, that is the training instances are not i.i.d.

5 Empirical Evaluation

We investigate our approaches by applying them to an email batch identification task. We compare the presented training methods with the iterative learning procedure for support vector machines with structured outputs by Finley and Joachims [13]. We explore the benefit of each approach and perform an error analysis.

In our experiments we use a slightly modified variant of the loss function based on the Rand index. Instead of normalizing over the number of all mails as in Equation 2 we use the number of emails in the current batch as normalization. That is, each wrong edge is weighted by the inverse size of its batch. The loss function 15 is linear in $\mathbf{z}$ and independent of the size of the batches, and thus better reflects the intuition about the quality of a batch detection method.

$$\Delta_N(\mathbf{y}^*, \mathbf{y}, j) = \sum_{k \neq j} \frac{[[\,[[y_j^* = y_k^*]] \neq [[y_j = y_k]]\,]]}{\sum_{k' \neq j}[[y_{k'}^* = y_k^*]]}. \tag{15}$$

The feature functions are simple pairwise indicators or measures, such as equality of sender or mimetype, difference of message length, edit-distance of the subject lines, cosine distance of TFIDF-vectors, or differences in letter-bigram-counts. Each wrong edge gets weighted by the inverse of the number of members of its corresponding batch, to even out the influences of large and small batches.

We evaluate our proposed methods on a set of 3000 emails, consisting of 2000 spam mails collected by an email service provider, 1000 non-spam mails from the public Enron corpus, and 500 newsletters. These mails were manually

grouped into batches, resulting in 136 batches with at average 17.7 emails and 598 remaining single mails. Our results are obtained through a cross validation procedure, where each test set contains a non-singular batch and is filled up with randomly drawn emails to a total size of 100 emails. The training data consist of nine sets of 100 emails each, sampled randomly from the remaining emails.

Each of the obtained models is applied to the test sets, using either the approximative clustering based on the linear program, the sequential clustering algorithm, or the greedy clustering algorithm by [13]. Figure 1 shows the experimental results of three of the training methods. The integrated learning problem is not tractable for this amount of training data.

In a second experiment, we split each training and each test set in two halves, resulting in 18 sets of 50 emails each for training. That is, the total number of training emails remains the same but the integrated learning problem becomes tractable. Figure 2 shows the results for this setting.

In both experiments, the LP-decoding strategy and the greedy clustering algorithm perform equally well. By contrast, the sequential clustering performs significantly worse in most of the cases according to a paired t-Test on a 5% confidence level. However, this loss in performance comes with a gain in execution time that is linear in the number of examples (see Table 2).

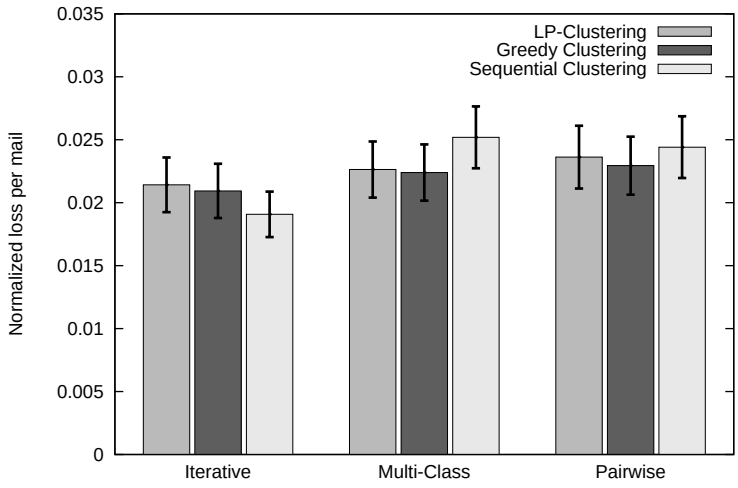


Fig. 1. Average loss and standard errors for $m = 100$.

Figure 3 details which fraction of the error is caused by the decoding and which by the learning algorithm. The dashed area indicates the error caused by the training method. We quantify this error by counting the number of different edges in the true and the predicted similarity matrix, respectively. The additional error of the subsequent decoding is indicated for all three decoding strategies.

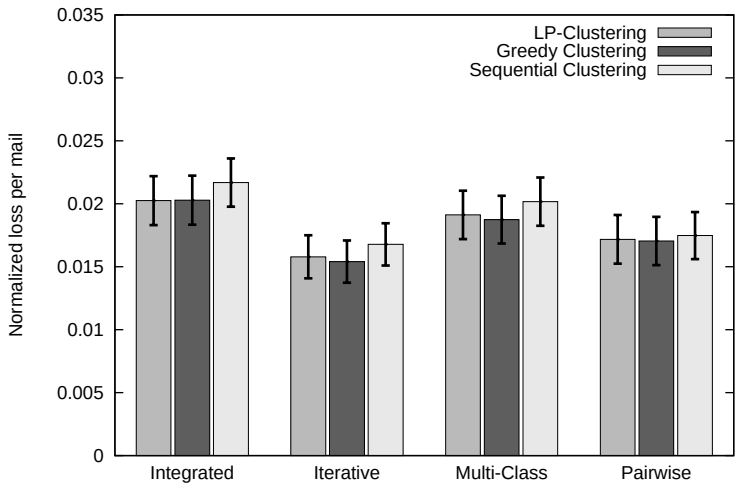


Fig. 2. Average loss and standard errors for $m = 50$.

Except for the sequential decoding, the multi-class optimization leads to correct clusterings that fulfill the transitivity constraints between triples of nodes. On the contrary, the solution of the pairwise optimization has the lowest error but fails to satisfy these transitivity constraints. Neither decoding strategy can compensate the errors.

Table 2. Execution time of the decoding strategies in seconds.

Window size m	25	50	100	200
LP-Clustering	$2.3 \cdot 10^{-1}$	$6.4 \cdot 10^{0}$	$4.0 \cdot 10^{2}$	
Greedy Clustering	$6.4 \cdot 10^{-4}$	$2.5 \cdot 10^{-3}$	$9.9 \cdot 10^{-3}$	$4.0 \cdot 10^{-2}$
Sequential Clustering	$1.5 \cdot 10^{-5}$	$2.9 \cdot 10^{-5}$	$5.6 \cdot 10^{-5}$	$1.1 \cdot 10^{-4}$

6 Conclusion

We devised three large margin approaches to supervised clustering of sequential data. The integrated approach has at least cubic execution time and can be solved directly for small training sets. Treating the problem as multi-class classification allowed us to use larger data sets. The pairwise classification approach is a rough but fast approximation of the original problem.

Experimental results were carried out on all combinations of learning algorithms and decoding strategies in our discourse area. The results showed that the LP-decoding performs equally well as the greedy algorithm presented in [13].

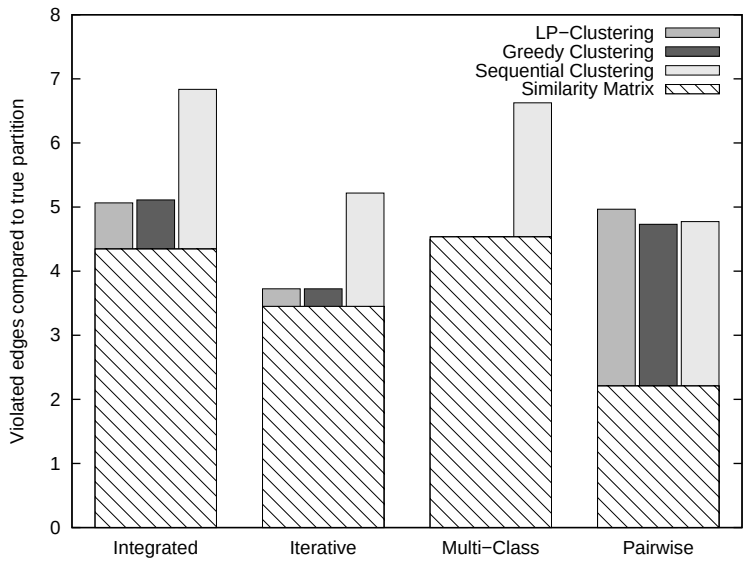


Fig. 3. Fraction of the loss induced by the learning algorithm (similarity matrix) and the decoding.

However, both methods are computationally expensive. The sequential decoding makes use of the sequential nature of the data and leads to slightly increased losses.

References

1. Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden Markov support vector machines. In *Proceedings of the International Conference on Machine Learning*, 2003.
2. Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, 2002.
3. Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.
4. Aharon Bar-Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall. Learning distance functions using equivalence relations. In *ICML '03: Proceedings of the Twentieth International Conference on Machine Learning*, 2003.
5. Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2003.
6. Sergey Brin, James Davis, and Hector García-Molina. Copy detection mechanisms for digital documents. In *Proceedings of the International Conference on Management of Data*, 1995.

7. Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003.
8. J. Cooper, A. Coden, and E. Brown. Detecting similar documents using salient terms. In *Proceedings of the International Conference on Information and Knowledge Management*, 2002.
9. J. Cooper, A. Coden, and E. Brown. A novel method for detecting similar documents. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, 2002.
10. Erik D. Demaine and Nicole Immorlica. Correlation clustering with partial information. In *Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 7th International Workshop on Randomization and Approximation Techniques in Computer Science*, 2003.
11. Dotan Emanuel and Amos Fiat. Correlation clustering – minimizing disagreements on arbitrary weighted graphs. *Lecture Notes in Computer Science*, 2832:208–220, 2003.
12. Michael Fink, Shai Shalev-Shwartz, Yoram Singer, and Shimon Ullman. Online multiclass learning by interclass hypothesis sharing. In *ICML '06: Proceedings of the 23nd international conference on Machine learning*, 2006.
13. Thomas Finley and Thorsten Joachims. Supervised clustering with support vector machines. In *Proceedings of the International Conference on Machine Learning*, 2005.
14. Aleksander Kolcz, Abdur Chowdhury, and Joshua Alspector. The impact of feature selection on signature-driven spam detection. In *Proceedings of the First Conference on Email and Anti-Spam*, 2004.
15. J. Lafferty, X. Zhu, and Y. Liu. Kernel conditional random fields: representation and clique selection. In *Proc. of the International Conference on Machine Learning*, 2004.
16. W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:622–626, 1971.
17. Chaitanya Swamy. Correlation clustering: maximizing agreements via semidefinite programming. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, 2004.
18. B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *Advances in Neural Information Processing Systems*, 2004.
19. Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: a large margin approach. In *Proceedings of the International Conference on Machine Learning*, 2005.
20. I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.
21. Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 577–584, San Francisco, CA, USA, 2001.
22. Eric P. Xing, Andrew Y. Ng, Michael I. Jordan, and Stuart Russell. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems*. The MIT Press, 2002.
23. Feng Zhou, Li Zhuang, Ben Y. Zhao, Ling Huang, Anthony D. Joseph, and John Kubiatowicz. Approximate object location and spam filtering on peer-to-peer systems. In *Middleware*, 2003.

Frequent Hypergraph Mining

Tamás Horváth[1,2], Björn Bringmann[3], and Luc De Raedt[3]

[1] Department of Computer Science III, University of Bonn, Germany
[2] Fraunhofer IAIS, Sankt Augustin, Germany
[3] Institute for Computer Science, Machine Learning Lab, University of Freiburg, Germany

Abstract. The class of frequent hypergraph mining problems is introduced which includes the frequent graph mining problem class and contains also the frequent itemset mining problem. We study the computational properties of different problems belonging to this class. In particular, besides negative results, we present practically relevant problems that can be solved in incremental-polynomial time. Some of our practical algorithms are obtained by reductions to frequent graph mining and itemset mining problems. Our experimental results in the domain of citation analysis show the potential of the framework on problems that have no natural representation as an ordinary graph.

1 Introduction

The field of data mining has studied increasingly expressive representations in the past few years. Whereas the original formulation of frequent pattern mining still employed itemsets [1], researchers have soon studied more expressive representations such as sequences and episodes (e.g., [11]), trees (e.g., [4]), and more recently, graph mining has become an important focus of research (e.g., [12, 13]). These developments have been motivated and accompanied by new and challenging application areas. Indeed, itemsets apply to basket-analysis, sequences and episodes to alarm monitoring, trees to document mining, and graph mining to applications in computational chemistry.

In this paper, we introduce the next natural step in this evolution: the mining of *labeled hypergraphs*. In a similar way that tree mining generalizes sequence mining, and graph mining generalizes tree mining, hypergraph mining is a natural generalization of graph mining. The presented framework is especially applicable to problem domains which do not have a natural representation as ordinary graphs. One such application is used in the experimental section of this paper. It is concerned with citation analysis, more specifically, with analyzing bibliographies of a set of papers. The bibliography of a paper can be viewed as a hypergraph, in which each author corresponds to a vertex and each paper to the hyperedge containing all authors of the paper. By mining for frequent subhypergraphs in the bibliographies of a set of papers (e.g. past KDD conference papers), one should be able to discover common citation patterns in a particular domain (such as SIGKDD). These patterns might then be employed in a recommender system that assists scientists while making bibliographies. A similar approach in a basket-analysis context allows one to represent the transactions over a specific period of time of *one family* as a hypergraph, where the products correspond to the vertices and the

transactions to the hyperedges. Mining such data could provide insight into the overall purchasing behavior of families.

The main contribution of this paper is the introduction of a general framework of mining frequent hypergraphs. The framework can be specialized in a number of different ways, according to the notion of the generalization relation employed as well as the type of hypergraphs considered. We consider different problems where the generalization relation is defined by *subhypergraph isomorphism*, study their computational properties, and present positive and negative results. More specifically, we show that there is no output-polynomial time algorithm for the frequent hypergraph mining problem even in the case of strong *structural* assumptions on the hyperedges. On the other hand, by restricting the functions labeling the vertices, we get positive results. Some of the results are obtained by employing reductions from frequent hypergraph mining problems to ordinary graph mining and itemset mining problems. We present also experiments in the above sketched citation analysis domain which indicate that these reductions can effectively be applied in practice. Essentially, we gathered the bibliographies of 5 SIGKDD, 30 SIGMOD, and 30 SIGGRAPH conferences and searched for frequent hypergraphs in each conference.

The rest of the paper is organized as follows: in Section 2, we introduce the necessary notions concerning hypergraphs and in Section 3, we define the problem class of frequent hypergraph mining. In Section 4, we study the frequent subhypergraph mining problem. In Section 5, we present some experiments using the citation analysis problem, and finally, in Section 6, we conclude and list some problems for future work. Due to space limitations, proofs are only sketched or even omitted in this short version.

2 Notions and Notations

We recall some basic notions and notations related to graphs and hypergraphs (see, e.g., [2, 5] for detailed introductions into these fields). For a set S and non-negative integer k, $[S]^k$ denotes the family of k-subsets of S, i.e., $[S]^k = \{S' \subseteq S : |S'| = k\}$.

Graphs and Hypergraphs An *(undirected) graph* G consists of a finite set V of *vertices* and a set $\mathcal{E} \subseteq [V]^2$ of *edges*. G is *bipartite* if G has a vertex 2-coloring, i.e., if V admits a partition into V_1 and V_2 such that $E \notin [V_1]^2 \cup [V_2]^2$ for every $E \in \mathcal{E}$. A *hypergraph* H is a pair $(V, \mathcal{E})$, where V is a finite set and $\mathcal{E}$ is a family of nonempty subsets of V such that $\bigcup_{E\in\mathcal{E}} E = V$. The elements of V and $\mathcal{E}$ are called *vertices* and *edges* (or *hyperedges*), respectively. H is *r-uniform* for some integer $r > 0$ if $\mathcal{E} \subseteq [V]^r$. The *rank* of H, denoted $r(H)$, is the cardinality of its largest hyperedge and the *size* of H, denoted size(H), is the number of hyperedges of H.

Note that ordinary undirected graphs without isolated vertices form a special case of hypergraphs, i.e., the class of 2-uniform hypergraphs. We note that every hypergraph $H = (V, \mathcal{E})$ can be represented by a *bipartite incidence* graph $B(H) = (V \cup \mathcal{E}, \mathcal{E}')$, where $\mathcal{E}' = \{\{v, E\} : v \in V, E \in \mathcal{E}, \text{ and } v \in E\}$.

Labeled Hypergraphs A *labeled hypergraph* is a triple $H = (V, \mathcal{E}, \lambda)$, where $(V, \mathcal{E})$ is a hypergraph, and λ, called *labeling function*, is a function mapping V to $\mathbb{N}$.[4] Unless

[4] We will only consider labeling functions defined on the vertex set because any hypergraph $H = (V, \mathcal{E}, \lambda)$ with $\lambda : V \cup \mathcal{E} \to \mathbb{N}$ satisfying $\lambda(v) \neq \lambda(E)$ for every $v \in V$ and $E \in \mathcal{E}$

otherwise stated, by hypergraphs (resp. graphs) we always mean labeled hypergraphs (resp. labeled graphs), and denote the set of vertices, the set of edges, and the labeling function of a hypergraph (resp. graph) H by V_H, $\mathcal{E}_H$, and λ_H, respectively. The set of all hypergraphs is denoted by $\mathcal{H}$ and $\mathcal{H}_r$ denotes the set of all r-uniform hypergraphs. For a hypergraph $H \in \mathcal{H}$ and subset $V' \subseteq V_H$, we denote the *multiset*[5] $\{\lambda_H(v) : v \in V'\}$ by $\lambda^H(V')$. A *path* connecting the vertices $u, v \in V_H$ is a sequence $E_1, \ldots, E_k$ of edges of H such that $u \in E_1$, $v \in E_k$, and $E_i \cap E_{i+1} \neq \emptyset$ for every $i = 1, \ldots, k-1$. A hypergraph is *connected* if there is a path between any pair of its vertices. The set of connected hypergraphs is denoted by $\mathcal{H}^c$. Clearly, $\mathcal{H}^c \subset \mathcal{H}$.

Injective Hypergraphs Depending on the labeling functions, in this paper we will consider two special classes of hypergraphs. A hypergraph $H \in \mathcal{H}$ is *node injective* if λ_H is injective, and it is *edge injective* whenever $\lambda^H(E) = \lambda^H(E')$ if and only if $E = E'$ for every $E, E' \in \mathcal{E}_H$. The sets of node and edge injective hypergraphs will be denoted by $\mathcal{H}^{\text{ni}}$ and $\mathcal{H}^{\text{ei}}$, respectively. Clearly, $\mathcal{H}^{\text{ni}} \subseteq \mathcal{H}^{\text{ei}} \subseteq \mathcal{H}$.

Hypergraph Isomorphism Let $H_1, H_2 \in \mathcal{H}$ be hypergraphs. H_1 and H_2 are called *isomorphic*, denoted by $H_1 \simeq H_2$, if there is a bijection $\varphi : V_{H_1} \to V_{H_2}$ such that φ preserves the labels, i.e., $\lambda_{H_1}(v) = \lambda_{H_2}(\varphi(v))$ for every $v \in V_{H_1}$, and φ preserves the hyperedges in both directions, i.e., for every $E \subseteq V_{H_1}$ it holds that $E \in \mathcal{E}_{H_1}$ if and only if $\{\varphi(v) : v \in E\} \in \mathcal{E}_{H_2}$. Throughout this paper, two hypergraphs H_1 and H_2 are considered to be the same if $H_1 \simeq H_2$.

Subhypergraphs A *subhypergraph* of a hypergraph $H \in \mathcal{H}$ is a hypergraph $H' \in \mathcal{H}$ satisfying $V_{H'} \subseteq V_H$, $\mathcal{E}_{H'} \subseteq \mathcal{E}_H$, and $\lambda_{H'}(v) = \lambda_H(v)$ for every $v \in V_{H'}$.

3 Frequent Hypergraph Mining

Many problems in data mining can be viewed as a special case of the problem of enumerating the elements of a *quasiordered* set[6], which satisfy some monotone property (see, e.g., [3, 9]). In this section, we define a new class of subproblems of this enumeration problem, the class of *frequent hypergraph mining problems*. In the next section, we then discuss the computational aspects of some problems belonging to this class. We start with the definition of a more general problem class.

The Frequent Pattern Mining Problem Class ($\mathcal{C}_{\text{FPM}}$): Each problem belonging to this class is given by a *fixed* triple $(\mathcal{L}_D, \mathcal{L}_P, \preccurlyeq)$, where $\mathcal{L}_D$ is a *transaction language*, $\mathcal{L}_P$ is a *pattern language*, and $\preccurlyeq$, called the *generalization relation*, is a quasi-order on $\mathcal{L}_D \cup \mathcal{L}_P$. For such a triple, the $(\mathcal{L}_D, \mathcal{L}_P, \preccurlyeq)$-FREQUENT-PATTERN-MINING problem is defined as follows: *Given* a finite set $\mathcal{D} \subseteq \mathcal{L}_D$ of *transactions* and an integer $t > 0$, called *frequency threshold*, *compute* the set $\mathcal{F}_{(\mathcal{L}_D, \mathcal{L}_P, \preccurlyeq)}(\mathcal{D}, t)$ of *frequent patterns* defined by

$$\mathcal{F}_{(\mathcal{L}_D, \mathcal{L}_P, \preccurlyeq)}(\mathcal{D}, t) = \{\varphi \in \mathcal{L}_P : |\{\tau \in \mathcal{D} : \varphi \preccurlyeq \tau\}| \geq t\} \ .$$

can be transformed into a hypergraph $H' = (V', \mathcal{E}', \lambda')$ with $V' = V \cup \{v_E : E \in \mathcal{E}\}$, $\mathcal{E}' = \{E \cup \{v_E\} : E \in \mathcal{E}\}$, and with $\lambda' : V' \to \mathbb{N}$ mapping every new vertex $v_E \in V' \setminus V$ to $\lambda(E)$ and every $v \in V$ to $\lambda(v)$.

[5] A *multiset* M is a pair (S, f), where S is a set and f defines the multiplicity of the elements of S in M, i.e., f is a function mapping S to the cardinal numbers greater than 0.

[6] A binary relation is a *quasiorder* (or *preorder*), if it is reflexive and transitive.

The transitivity of $\preccurlyeq$ implies that frequency is a monotone property, i.e., for every $\varphi, \theta \in \mathcal{L}_P$ it holds that $\theta \in \mathcal{F}_{(\mathcal{L}_D,\mathcal{L}_P,\preccurlyeq)}(\mathcal{D}, t)$ whenever $\varphi \in \mathcal{F}_{(\mathcal{L}_D,\mathcal{L}_P,\preccurlyeq)}(\mathcal{D}, t)$ and $\theta \preccurlyeq \varphi$.

We now define two subclasses of $\mathcal{C}_{\mathrm{FPM}}$ by restricting the transaction and pattern languages to hypergraphs and graphs, respectively.

The Frequent Hypergraph Mining Problem Class ($\mathcal{C}_{\mathrm{FHM}}$): It consists of the set of $(\mathcal{L}_D, \mathcal{L}_P, \preccurlyeq)$-FREQUENT-PATTERN-MINING problems satisfying $\mathcal{L}_D, \mathcal{L}_P \subseteq \mathcal{H}$.

The Frequent Graph Mining Problem Class ($\mathcal{C}_{\mathrm{FGM}}$): It is the set of $(\mathcal{L}_D, \mathcal{L}_P, \preccurlyeq)$-FREQUENT-PATTERN-MINING problems satisfying $\mathcal{L}_D, \mathcal{L}_P \subseteq \mathcal{H}_2$ (i.e., they are sets of labeled graphs).

Clearly, $\mathcal{C}_{\mathrm{FGM}} \subsetneq \mathcal{C}_{\mathrm{FHM}} \subsetneq \mathcal{C}_{\mathrm{FPM}}$. Furthermore, the *frequent itemset mining problem* [1] belongs to $\mathcal{C}_{\mathrm{FPM}}$; for this problem we have $\mathcal{L}_D = \mathcal{L}_P = \{X \subset \mathbb{N} : |X| < \infty\}$ and $\preccurlyeq$ is the subset relation. In fact, the frequent itemset mining problem is contained by $\mathcal{C}_{\mathrm{FHM}}$. Indeed, this problem can be considered as the $(\mathcal{H}_1^{\mathrm{ni}}, \mathcal{H}_1^{\mathrm{ni}}, \preccurlyeq)$-FREQUENT-HYPERGRAPH-MINING problem, where $\preccurlyeq$ is the subhypergraph relation and the transaction and pattern languages are the set of 1-uniform node injective hypergraphs.

To sketch the relation among frequent pattern mining problems, we need the notion of *polynomial reduction*. More precisely, let $P_1 = (\mathcal{L}_{D,1}, \mathcal{L}_{P,1}, \preccurlyeq_1)$ and $P_2 = (\mathcal{L}_{D,2}, \mathcal{L}_{P,2}, \preccurlyeq_2)$ be frequent pattern mining problems, and $I_1 = (\mathcal{D}_1, t_1)$ and $I_2 = (\mathcal{D}_2, t_2)$ be instances of P_1 and P_2, respectively. Then I_1 is *polynomially equivalent* to I_2 if there is a function $f : \mathcal{L}_{P,1} \to \mathcal{L}_{P,2}$ such that

(i) f is a bijection between $\mathcal{F}_{(\mathcal{L}_{D,1},\mathcal{L}_{P,1},\preccurlyeq_1)}(\mathcal{D}_1, t_1)$ and $\mathcal{F}_{(\mathcal{L}_{D,2},\mathcal{L}_{P,2},\preccurlyeq_2)}(\mathcal{D}_2, t_2)$ and
(ii) the inverse of f on $\mathcal{F}_{(\mathcal{L}_{D,2},\mathcal{L}_{P,2},\preccurlyeq_2)}(\mathcal{D}_2, t_2)$ can be computed in polynomial time.

The definition implies that $\varphi \in \mathcal{L}_{P,1}$ is frequent for I_1 if and only if $f(\varphi) \in \mathcal{L}_{P,2}$ is frequent for I_2. Using the notion of polynomial equivalence, we say that P_1 is *polynomially reducible* to P_2 if there is a function g from the set of instances of P_1 to the set of instances of P_2 such that

(i) I is polynomially equivalent to $g(I)$ for every $I \in P_1$ and
(ii) g can be computed in polynomial time.

Thus, if P_1 is polynomial-time reducible to P_2 then any enumeration algorithm solving P_2 can be used to solve P_1.

The *parameter* of a $(\mathcal{L}_D, \mathcal{L}_P, \preccurlyeq)$-FREQUENT-HYPERGRAPH-MINING problem formulated above is the *size* of $\mathcal{D}$ defined by

$$\mathrm{size}(\mathcal{D}) = \max\left\{\sum_{H\in\mathcal{D}} \mathrm{size}(H), \max_{H\in\mathcal{D}} r(H)\right\} \ .$$

Note that the size of the output, i.e. the set to be enumerated, can be exponential in the size of the input. Because in such cases, it is impossible to compute them in time polynomial only in the size of the input, we investigate whether the enumeration problems can be solved in *incremental polynomial time* or at least in *output-polynomial time* (or *polynomial total time*) (see, e.g., [10]). In the first, more restrictive case, the algorithm

is required to list the first N elements of the output in time polynomial in the *combined size* of the input and the set of these N elements. In the second, more liberal case, the algorithm has to solve the problem in time polynomial in the combined size of the input and the *entire* set to be enumerated. Note that the class of output-polynomial time algorithms properly entails the class of incremental polynomial time algorithms.

To close this section, we note that several frequent hypergraph mining problems, even frequent graph mining problems, cannot be solved in output-polynomial time. In Theorem 1 below we present such a hard problem.

Theorem 1 *Let $\mathcal{L}_D \subseteq \mathcal{H}_2$ and let $\mathcal{L}_P \subseteq \mathcal{H}_2$ be the set of complete graphs such that every vertex of every graph in $\mathcal{L}_D \cup \mathcal{L}_P$ is labeled by the same symbol, say 1, and let $\preccurlyeq$ be the homomorphism $\preccurlyeq_h$ between labeled graphs*[7]*. Then, unless P = NP, the $(\mathcal{L}_D, \mathcal{L}_P, \preccurlyeq_h)$-*FREQUENT-GRAPH-MINING *problem cannot be solved in output polynomial time.*

Proof (sketch). Let $G = (V, \mathcal{E})$ be an unlabeled graph and let G' be the labeled graph obtained from G by assigning 1 to each vertex of G. Then, for $\mathcal{D} = \{G'\}$ and $t = 1$, we have that G has a clique of size k if and only if there is a $C \in \mathcal{F}_{(\mathcal{L}_D, \mathcal{L}_P, \preccurlyeq_h)}(\mathcal{D}, t)$ with k vertices. Since $|\mathcal{F}_{(\mathcal{L}_D, \mathcal{L}_P, \preccurlyeq_h)}(\mathcal{D}, t)| \leq |V|$, the $(\mathcal{L}_D, \mathcal{L}_P, \preccurlyeq_h)$-FREQUENT-GRAPH-MINING problem cannot be computed in output polynomial time (unless P = NP, of course), as otherwise the NP-complete maximum clique problem [8] could be decided in polynomial time by computing the largest pattern in $|\mathcal{F}_{(\mathcal{L}_D, \mathcal{L}_P, \preccurlyeq_h)}(\mathcal{D}, t)|$.

4 Frequent Subhypergraph Mining

By Theorem 1, the class $\mathcal{C}_{\mathrm{FGM}}$, and thus the more general class $\mathcal{C}_{\mathrm{FHM}}$ as well, contains problems that cannot be solved in output polynomial time (unless P = NP, of course). This negative result raises the challenge of identifying practically relevant and tractable problems belonging to $\mathcal{C}_{\mathrm{FHM}}$. In this section, we take a first step towards this direction by considering the problem of frequent hypergraph mining w.r.t. *subhypergraph isomorphism*. This problem, called *frequent subhypergraph mining*, is a natural problem of the frequent hypergraph mining problem class $\mathcal{C}_{\mathrm{FHM}}$ and can be applied to many practical problems. In Section 5, we will employ this setting to tackle the citation analysis problem sketched in the introduction.

We start with the definition of the generalization relation used in this section. Let $H_1, H_2 \in \mathcal{H}$. H_1 can be embedded into H_2 by *subhypergraph isomorphism*, denoted by $H_1 \preccurlyeq_i H_2$, if H_2 has a subhypergraph isomorphic to H_1. Note that $\preccurlyeq_i$ generalizes the notion of subgraph isomorphism between ordinary labeled graphs to hypergraphs. Since $\preccurlyeq_i$ is a partial order on $\mathcal{H}$, it is a generalization relation on every subset of $\mathcal{H}$. Using $\preccurlyeq_i$, we consider the following two problems of $\mathcal{C}_{\mathrm{FHM}}$:

(i) The $(\mathcal{H}, \mathcal{H}, \preccurlyeq_i)$-FREQUENT-HYPERGRAPH-MINING problem called the *frequent subhypergraph mining problem* and
(ii) the $(\mathcal{H}, \mathcal{H}^c, \preccurlyeq_i)$-FREQUENT-HYPERGRAPH-MINING problem called the *frequent connected subhypergraph mining problem*.

[7] A *homomorphism* from a hypergraph $H_1 \in \mathcal{H}$ to a hypergraph $H_2 \in \mathcal{H}$, denoted $H_1 \preccurlyeq_h H_2$, is a function $\varphi : V_{H_1} \to V_{H_2}$ preserving the labels and edges.

Algorithm 1 FREQUENT SUBHYPERGRAPH MINING

Require: instance $(\mathcal{D}, t)$
Ensure: $\mathcal{F}_{(\mathcal{H},\mathcal{H},\preccurlyeq_i)}(\mathcal{D}, t)$

1: $\mathcal{F} := \emptyset$
2: $\mathcal{B}_\mathcal{D} := \{LB(H) : H \in \mathcal{D}\}$
3: Compute a next t-frequent bipartite subgraph B of the set $\mathcal{B}_\mathcal{D}$ if it exists; otherwise **return** $\mathcal{F}$
4: **if** B corresponds to some hypergraph H_B **then**
$\mathcal{F} := \mathcal{F} \cup \{H_B\}$
5: **goto** 3

4.1 A Naïve Algorithm

Using the bipartite graph representation of hypergraphs, in this section we present a naïve algorithm solving the frequent subhypergraph mining problem. For an instance $(\mathcal{D}, t)$ of this problem, let $n \in \mathbb{N}$ be an upper bound on the labels occurring in the hypergraphs of $\mathcal{D}$ and let μ be an injection assigning an integer greater than n to every finite multiset of $\mathbb{N}$.

For a hypergraph $H \in \mathcal{H}$, let $LB(H) \in \mathcal{H}_2$ be the (labeled) *bipartite* graph such that

(i) $(V_{LB(H)}, \mathcal{E}_{LB(H)})$ is the unlabeled bipartite incidence graph of the unlabeled hypergraph $(V_H, \mathcal{E}_H)$, and
(ii) for every $v \in V_{LB(H)} = V_H \cup \mathcal{E}_H$,

$$\lambda_{LB(H)}(v) = \begin{cases} \lambda_H(v) & \text{if } v \in V_H \\ \mu(\lambda^H(v)) & \text{otherwise (i.e., } v \in \mathcal{E}_H\text{)} . \end{cases}$$

Clearly, a subgraph G of $LB(H)$ represents a subhypergraph of H if and only if each vertex of G corresponding to a hyperedge $E \in \mathcal{E}_H$ is connected with exactly $|E|$ vertices in G. Using the above transformation and considerations, the set $\mathcal{F}_{(\mathcal{H},\mathcal{H},\preccurlyeq_i)}(\mathcal{D}, t)$ of t-frequent subhypergraphs for the instance $(\mathcal{D}, t)$ can be computed by Algorithm 1.

Due to space limitations, we omit a detailed analysis of Algorithm 1 that does not work in output polynomial time in the worst case. We note, however, that even this naïve algorithm proved to be effective in time on the citation analysis domain (cf. Section 5).

4.2 Negative Results

In this section we investigate further problems obtained by *structural* restrictions hoping to identify tractable fragments of the frequent subhypergraph mining problem. Unfortunately, we have not been able to find any interesting positive result in this way. This is because, as indicate the negative results of this section, the problem remains hard even for very restricted hypergraph classes. Without proof, in Theorem 2 below we state that even for 2-uniform hypergraphs (i.e., ordinary graphs), the frequent connected subhypergraph mining problem is intractable in output-polynomial time. Since this is one

of the most frequently considered frequent graph mining problems, the negative result below may be of interest in itself.

Theorem 2 *If $P \neq NP$, there is no output-polynomial time algorithm solving the frequent connected subhypergraph mining problem even in the case of 2-unary hypergraphs (i.e., ordinary graphs).*

As a further restriction, we consider the frequent subhypergraph mining problem restricted to acyclic hypergraphs [7] because several NP-hard problems on hypergraphs become polynomial for acyclic hypergraphs. A hypergraph $H \in \mathcal{H}$ is *α-acyclic* if one can remove all of its vertices and edges by deleting repeatedly either an edge that is empty or is contained by another edge, or a vertex contained by at most one edge [14]. Note that α-acyclicity is not a hereditary property, that is, α-acyclic hypergraphs may have subhypergraphs that are not α-acyclic. Consider for example the hypergraph $H \in \mathcal{H}$ such that $\mathcal{E}_H = \{\{a,b\},\{b,c\},\{a,c\},\{a,b,c\}\}$. While H is α-acyclic, its subhypergraph obtained by removing the edge $\{a,b,c\}$ is not α-acyclic. To overcome this anomaly, the following proper subclass of α-acyclic hypergraphs is introduced in [7]: An α-acyclic hypergraph is *β-acyclic*, if each of its subhypergraphs is also α-acyclic. Note that forests are 2-uniform β-acyclic hypergraphs.

Even for connected subhypergraphs of 3-uniform β-acyclic hypergraphs, we have a negative result. Let $\mathcal{B}_3$ denote the set of 3-uniform β-acyclic hypergraphs.

Proposition 3 *Given a finite set $\mathcal{D} \subseteq \mathcal{B}_3$ and integer $t > 0$, deciding whether $H \in \mathcal{F}_{(\mathcal{B}_3,\mathcal{H}^c,\preccurlyeq_i)}(\mathcal{D},t)$ is NP-hard.*

Proof (sketch). Using a polynomial reduction from the *subforest isomorphism* problem[8], one can show that deciding subhypergraph isomorphism between 3-uniform, connected, β-acyclic hypergraphs is NP-hard. This implies the statement.

Proposition 3 above indicates that for the frequent subhypergraph mining problem, the usual frequent pattern mining approaches (such as the level-wise one) will not work in incremental polynomial time (unless $P = NP$) because they repeatedly test whether candidate patterns satisfy the frequency threshold (see, e.g., [9]).

4.3 Tractable Cases

In contrast to the approach discussed in Section 4.2 above, in this section we consider further special cases of the frequent subhypergraph mining problem that are obtained by making assumptions on the *labeling functions* of the transaction hypergraphs. We first consider the problem for *node injective* hypergraphs, i.e., where the labeling functions are injective. We show that for this case, the frequent subhypergraph mining problem is polynomially reducible to the frequent itemset mining problem and hence, it can be solved in incremental-polynomial time [1]. We then generalize this positive result to edge injective hypergraphs, i.e., to hypergraphs not containing two different hyperedges that are mapped to the same multiset by the labeling function. Although node injective

[8] Given a forest F and a tree T, decide whether T has a subgraph isomorphic to F. This problem is known to be NP-complete [8].

hypergraphs are a special case of edge injective hypergraphs, we discuss the two cases separately because node injective hypergraphs can be used to model many practical problems and they permit a simplified algorithmic approach.

Node Injective Hypergraphs As mentioned above, many practical data mining problems can be modeled by node injective hypergraphs, i.e., by hypergraphs from $\mathcal{H}^{\text{ni}}$. Such applications include problem domains consisting of a finite set of objects (vertices) with a unique identifier. For node injective hypergraphs, we consider the $(\mathcal{H}^{\text{ni}}, \mathcal{H}^{\text{ni}}, \preccurlyeq_i)$-FREQUENT-HYPERGRAPH-MINING problem which is a special case of the frequent subhypergraph mining problem.

As an example of a practical application of this problem, we consider the *citation analysis* task mentioned in the introduction (cf. also Section 5): *Given* a set $\mathcal{D}$ of articles and a frequency threshold $t > 0$, *print* each family $\mathcal{F}$ of groups of authors satisfying the following property: there exists a subset $\mathcal{D}' \subseteq \mathcal{D}$ of articles of cardinality at least t such that for every group $F \in \mathcal{F}$ of authors and for every article $D \in \mathcal{D}'$ it holds that D cites some article written by (exactly) the authors belonging to F. In this enumeration problem, we can assign a unique non-negative integer to each author, whose papers are cited by at least one article in $\mathcal{D}$. We can use the *node injective hypergraph representation* of a paper's bibliography defined as follows. For each author cited in the bibliography, introduce a vertex and label it by the integer assigned to the author. Furthermore, for each cited work add a hyperedge E to the set of hyperedges, where E consists of the vertices representing the cited work's authors. Clearly, the hypergraph obtained in this way is always node injective. Our database $\mathcal{D}$ is a set of such node injective hypergraphs.

Theorem 4 below states that for node injective hypergraphs, the frequent subhypergraph mining problem is polynomially reducible to the frequent itemset mining problem. We recall that the frequent itemset mining problem can be considered as a problem belonging to the class $\mathcal{C}_{\text{FHM}}$ (cf. Section 3). Notice that in the theorem below, subhypergraphs may be non-connected. The theorem is based on the fact that for every node injective hypergraphs $H_1, H_2 \in \mathcal{H}^{\text{ni}}$, $H_1 \preccurlyeq_i H_2$ if and only if for every $E_1 \in \mathcal{E}_{H_1}$, there is a hyperedge $E_2 \in \mathcal{E}_{H_2}$ such that $\lambda^{H_1}(E_1) = \lambda^{H_2}(E_2)$, i.e.,

$$H_1 \preccurlyeq_i H_2 \iff \{\lambda^{H_1}(E) : E \in \mathcal{E}_{H_1}\} \subseteq \{\lambda^{H_2}(E) : E \in \mathcal{E}_{H_2}\} \ .$$

Note that the above equivalence implies that $\preccurlyeq_i$ can be decided efficiently for node injective hypergraphs.

Theorem 4 *The frequent subhypergraph mining problem for node injective hypergraphs is polynomially reducible to the frequent itemset mining problem.*

Due to space limitation, we omit the proof of the theorem which is based on considering the set of vertex labels of a hyperedge as an item for every hyperedge occurring in the transaction hypergraphs. Combining the above theorem with the results of [1], we have the following result on listing frequent subhypergraphs for node injective hypergraphs.

Corollary 5 *The frequent subhypergraph mining problem for node injective hypergraphs can be solved in incremental polynomial time.*

Algorithm 2 MINING EDGE INJECTIVE HYPERGRAPHS

Require: finite set $\mathcal{D} \subseteq \mathcal{H}^{\text{ei}}$ and integer $t > 0$
Ensure: $\mathcal{F}_{(\mathcal{H}^{\text{ei}}, \mathcal{H}^{\text{ei}}, \preccurlyeq_i)}(\mathcal{D}, t)$

1: $X := \bigcup_{H \in \mathcal{D}} \{\lambda^H(E) : E \in \mathcal{E}_H\}$
2: $F := \emptyset$
3: $k := 0$
4: **while** $k = 0 \vee L_k \neq \emptyset$ **do**
5: $k := k + 1$
6: $C_k := \begin{cases} X & \text{if } k = 1 \\ \{Y_1 \cup Y_2 \in [X]^k : Y_1, Y_2 \in L_{k-1}\} & \text{otherwise} \end{cases}$
7: $L_k := \emptyset$
8: **forall** $X' \in C_k$ **do**
9: $Q := \emptyset$
10: **forall** $H \in \mathcal{D}$ **do**
11: **if** $\exists H' \in \mathcal{H}$ s.t. $X' = \{\lambda^{H'}(E) : E \in \mathcal{E}_{H'}\}$ **then**
12: **if** $\exists (H'', f) \in Q$ s.t. $H'' \simeq H'$ **then**
13: change (H'', f) in Q to $(H'', f + 1)$
14: **else** $Q := Q \cup \{(H', 1)\}$
15: **endfor**
16: flag := TRUE
17: **forall** $(H, f) \in Q$ s.t. $f \geq t$ **do**
18: $F := F \cup \{H\}$
19: **if** flag **then**
20: $L_k := L_k \cup \{X'\}$
21: flag := FALSE
22: **endif**
23: **endfor**
24: **endfor**
25: **endwhile**
26: **return** F

Edge Injective Hypergraphs We now generalize the previous positive result to *edge injective hypergraphs*. Since a hypergraph now may contain two vertices with the same label, a family of multisets of labels doesn't define a hypergraph uniquely. Hence, a reduction to frequent itemset mining is not applicable to this case.

Theorem 6 *The frequent subhypergraph mining problem for edge injective hypergraphs can be solved in incremental polynomial time.*

Proof (sketch). Due to space limitations, we only sketch the proof. We first note that subhypergraph isomorphism between edge injective hypergraphs can be decided in polynomial time. To compute the set of t-frequent hypergraphs, we use an Apriori-like algorithm given in Algorithm 2.

In line 1 of the algorithm, X is initialized as the set of multisets corresponding to the edges in the transaction hypergraphs. In C_k (line 6), we compute a family of candidate sets of multisets. Each set in C_k consists of k multisets. For a set X' in C_k (see the

Table 1. Datasets used. We list the total number of papers in the proceedings and the number of authors occurring in the reference lists of the corresponding papers.

dataset	years	papers	authors
KDD	99-04	499	6966
SIGMOD	74-04	1404	11984
SIGGRAPH	74-04	1519	13192

loop starting at line 8), we check for every $H \in \mathcal{D}$ whether H has a subhypergraph H' such that the set of multisets defined by the edges of H' is equal to X'. Since edges are invectively labeled, H' must contain exactly k hyperedges. If H has such a subhypergraph H' then we check whether we have already found another hypergraph in the database which has a subhypergraph isomorphic to H'. If yes, we increment the counter of this subhypergraph (line 13); otherwise we add H' with frequency 1 to the set Q (line 14). In the loop (17–23) we update the set of frequent hypergraphs and L_k. One can show that this algorithm works in incremental polynomial time.

5 Experimental Evaluation

In this section, we evaluate our methods on the citation analysis problem discussed earlier. Three bibliographic datasets, the KDD, SIGMOD, and SIGGRAPH, were constructed from the ACM Digital Library[9]. They correspond to the set of all reference lists of papers found in the proceedings of the respective conferences. The characteristics of the datasets are listed in Table 1.

Each paper was represented as a hypergraph, as described in Section 4.3. The resulting hypergraphs are *node injective*, and in almost all cases, also disconnected. Because most existing graph miners only consider *connected* graphs, we added one special hyperedge to each paper, which connects all authors cited in that paper.

We performed experiments with the naïve algorithm based on reduction to frequent bipartite graph mining (cf. Section 4.1), as well as with the reduction to frequent itemset mining (cf. Section 4.3). All experiments were run on a workstation, running Suse Linux 9.2, 3.2 GHz, 2GB of RAM. As graph miner, we employed Siegfried Nijssens' GASTON [12] and as itemset miner, Bart Goethals' implementation[10] of Apriori. Because we did not employ a specialized hypergraph or graph miner, the data had to be pre- and post-processed. We used several `perl`-scripts to realize this. The pre- and post-processing steps run in time linear in the number of hypergraphs.

5.1 Experimental Results

The number of frequent patterns for different frequency thresholds for the three datasets is given in Figure 1. The runtime of the itemset miner was always below 0.1 seconds. Different from that, the naïve algorithm required much higher runtimes; 833.5, 16.8,

[9] `http://www.acm.org/`
[10] *http://www.cs.helsinki.fi/u/goethals/software/*

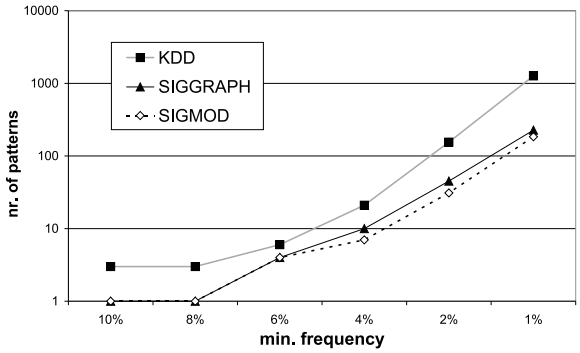


Fig. 1. Number of frequent patterns for different frequency thresholds.

and 9.9 seconds for KDD, SIGGRAPH, and SIGMOD, respectively. These higher runtimes are essentially due to the problem that only a fraction of the frequent bipartite graphs correspond in fact to subhypergraphs.

Despite the difference between the runtimes of the two approaches, our experiments clearly indicate that both reductions can be quite effective in practice.

6 Conclusion and Further Research

The problem class $\mathcal{C}_{\mathrm{FHM}}$ of frequent hypergraph mining was introduced. It forms a natural extension of traditional frequent itemset and graph mining. Several problems of $\mathcal{C}_{\mathrm{FHM}}$ were studied and positive and negative complexity results were obtained. Central to our results is the use of reductions from hypergraph mining problems to itemset and graph mining problems. These reductions are not only of theoretical interest, but allow us to set up a number of frequent hypergraph mining experiments even though we have not implemented a frequent hypergraph mining system. To our knowledge, the use of such reductions in data mining is new. Indeed, it is much more common in data mining to spend a lot of time and effort to develop new systems, even though they are sometimes only variants of existing ones. In our first step of studying some problems of $\mathcal{C}_{\mathrm{FHM}}$, we deliberately did not follow this common methodology, because there are many problems of $\mathcal{C}_{\mathrm{FHM}}$ that are interesting (which implies the need for implementing many variants and optimizations), and also, because we wanted to see how far the reductions would bring us. The experiments clearly indicate that - at least for the citation analysis problems studied - reductions can be quite effective in practice. In addition, these experiments provide evidence that frequent hypergraph mining is indeed a useful generalization of frequent itemset and graph mining and is likely to yield many interesting applications. Finally we list some open questions.

(i) One of the challenges is to identify further problems of $\mathcal{C}_{\mathrm{FHM}}$ that are enumerable in incremental or at least in output-polynomial time.
(ii) Besides subhypergraph isomorphism, it would be interesting to investigate frequent hypergraph mining problems, where the generalization relation is defined by (constrained) *homomorphisms*.

(iii) Since many problems of $\mathcal{C}_{\mathrm{FHM}}$ can be reduced to frequent graph mining in bipartite graphs, it would be interesting to develop frequent graph mining algorithms specific to bipartite graphs.
(iv) The work on frequent hypergraph mining can be related to multi-relational data mining [6], where each instance consists of multiple tuples over multiple tables in a relational database. Multi-relational data mining techniques have been applied to graph mining problems. Hence, the question arises if they are also applicable to hypergraph mining, and vice versa.

Acknowledgments

The authors thank Mario Boley and Stefan Wrobel for useful comments. Tamás Horváth was partially supported by the DFG project (WR 40/2-2) *Hybride Methoden und Systemarchitekturen für heterogene Informationsräume*.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast Discovery of Association Rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, Cambridge, MA, 1996.
2. C. Berge. *Hypergraphs*. North Holland Mathematical Library, Vol. 445. Elsevier, Amsterdam, 1989.
3. E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan, and K. Makino. Dual Bounded Hypergraphs: A Survey. In *Proc. of the 2nd SIAM Conference on Data Mining*, pages 87–98, 2002.
4. Y. Chi, R. R. Muntz, S. Nijssen, and J. N. Kok. Frequent Subtree Mining - An Overview. *Fundamenta Informaticae*, 66(1-2):161–198, 2005.
5. R. Diestel. *Graph Theory*. Springer, New York, 2nd edition, 2000.
6. S. Dzeroski and N. Lavrac, editors. *Relational Data Mining*. Springer, New York, 2002.
7. R. Fagin. Degrees of Acyclicity for Hypergraphs and Relational Database Schemes. *Journal of the ACM*, 30(3):514–550, 1983.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to NP-Completeness*. Freeman, San Francisco, CA, 1979.
9. D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharm. Discovering All Most Specific Sentences. *ACM Trans. on Database Systems*, 28(2):140–174, 2003.
10. D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On Generating All Maximal Independent Sets. *Information Processing Letters*, 27(3):119–123, 1988.
11. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.
12. S. Nijssen and J. N. Kok. A Quickstart in Frequent Structure Mining Can Make a Difference. In *Proc. of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 647–652. ACM Press, New York, NY, 2004.
13. X. Yan and J. Han. Gspan: Graph-based Substructure Pattern Mining. In *Proc. of the 2002 IEEE International Conference on Data Mining*, pages 721–724. IEEE Computer Society, 2002.
14. C. T. Yu and M. Z. Ozsoyoglu. An Algorithm for Tree-Query Membership of a Distributed Query. In *Proc. of Computer Software and Applications Conference*, pages 306–312. IEEE Computer Society, 1979.

Frequent Subgraph Mining in Outerplanar Graphs*

Tamás Horváth[1,2], Jan Ramon[3], and Stefan Wrobel[2,1]

[1] Dept. of Computer Science III, University of Bonn, Germany
[2] Fraunhofer IAIS, Sankt Augustin, Germany
[3] Dept. of Computer Science, Katholieke Universiteit Leuven, Belgium

Abstract. In recent years there has been an increased interest in frequent pattern discovery in large databases of graph structured objects. While the frequent connected subgraph mining problem for tree datasets can be solved in incremental polynomial time, it becomes intractable for arbitrary graph databases. Existing approaches have therefore resorted to various heuristic strategies and restrictions of the search space, but have not identified a practically relevant tractable graph class beyond trees. In this paper, we define the class of so called *tenuous outerplanar graphs*, a strict generalization of trees, develop a frequent subgraph mining algorithm for tenuous outerplanar graphs that works in incremental polynomial time, and evaluate the algorithm empirically on the NCI molecular graph dataset.

1 Introduction

The discovery of *frequent patterns* in a database is one of the central tasks considered in data mining. In addition to be interesting in their own right, frequent patterns can also be used as features for predictive data mining tasks (see, e.g., [5]). For a long time, work on frequent pattern discovery has concentrated on relatively simple notions of patterns and elements in the database as they are typically used for the discovery of association rules (simple sets of atomic items). In recent years, however, due to the significance of application areas such as the analysis of chemical molecules or graph structures in the WWW, there has been an increased interest in algorithms that can perform frequent pattern discovery in databases of structured objects such as *trees* or arbitrary *graphs*.

While the frequent pattern problem for trees can be solved in *incremental polynomial time*, i.e., in time polynomial in the *combined size* of the input and the set of frequent tree patterns *so far* computed, the frequent pattern problem for graph structured databases in the general case cannot be solved in *output polynomial time*, i.e., in time polynomial in the *combined size* of the input and the set of *all* frequent patterns. Existing approaches to frequent pattern discovery for graphs have therefore resorted to various heuristic strategies and restrictions of the search space (see, e.g., [4, 5, 8, 17]), but have not identified a practically relevant tractable graph class beyond trees.

In this paper, we define the class of so called *tenuous outerplanar graphs*, which is the class of planar graphs that can be embedded in the plane in such a way that all of its vertices lie on the outer boundary, i.e. can be reached from the outside without crossing any edges, and which have a fixed limit on the number of inside diagonal edges. This

* A longer version of this paper has been accepted to ACM SIGKDD'06.

class of graphs is a strict generalization of trees, and is motivated by the kinds of graphs actually found in practical applications. In fact, in one of the popular graph mining data sets, the NCI data set[4], 94.3% of all elements are tenuous outerplanar graphs. We develop an incremental polynomial time algorithm for enumerating frequent tenuous outerplanar graph patterns.

Our approach is based on a canonical string representation of outerplanar graphs which may be of interest in itself, and further algorithmic components for mining frequent biconnected outerplanar graphs and candidate generation in an Apriori style algorithm. To map a pattern to graphs in the database, we define a special notion of *block and bridge preserving* (BBP) subgraph isomorphism, which is motivated by application and complexity considerations, and show that it is decidable in polynomial time for outerplanar graphs. We note that for trees, which form a special class of outerplanar graphs, BBP subgraph isomorphism is equivalent to subtree isomorphism. Thus, BBP subgraph isomorphism generalizes subtree isomorphism to graphs, but is at the same time more specific than subgraph isomorphism. Since in many applications, subgraph isomorphism is a non-adequate matching operator (e.g., when pattern matching is required to preserve certain type of fragments in molecules), by considering BBP subgraph isomorphism we take a first step towards studying the frequent graph mining problem w.r.t. non-standard matching operators as well. Beside complexity results, we present also empirical results which show that the favorable theoretical properties of the algorithm and pattern class also translate into efficient practical performance.

The paper is organized as follows. In Sections 2 and 3, we define the necessary notions and the problem setting for this work, respectively. Section 4 describes our algorithm for mining frequent tenuous outerplanar graphs. Section 5 contains our experimental evaluation and finally, Section 6 concludes and discusses some open problems. Due to space limitations, proofs are omitted in this short version.

2 Preliminaries

We recall some notions related to graphs [7]. An *undirected graph* is a pair (V, E), where $V \neq \emptyset$ is a finite set of *vertices* and $E \subseteq \{e \subseteq V : |e| = 2\}$ is a set of *edges*. A *labeled undirected graph* is a quadruple (V, E, Σ, λ), where (V, E) is an undirected graph, $\Sigma \neq \emptyset$ is a finite set of *labels* associated with some total order, and $\lambda : V \cup E \to \Sigma$ is a function assigning a label to each element of $V \cup E$. Unless otherwise stated, in this paper by graphs we always mean *labeled undirected* graphs and denote the set of vertices, the set of edges, and the labeling function of a graph G by $V(G)$, $E(G)$, and λ_G, respectively. Let G and G' be graphs. G' is a *subgraph* of G, if $V(G') \subseteq V(G)$, $E(G') \subseteq E(G)$, and $\lambda_{G'}(x) = \lambda_G(x)$ for every $x \in V(G') \cup E(G')$. For a vertex $v \in V(G)$, $N(v)$ denotes the set of vertices of G connected by an edge with v.

A graph G is *connected* if there is a path between any pair of its vertices; it is *biconnected* if for any two vertices u and v of G, there is a simple cycle containing u and v. A *block* (or *biconnected component*) of a graph is a maximal subgraph that is biconnected. Edges not belonging to blocks are called *bridges*. The definitions imply

[4] http://cactus.nci.nih.gov/

that the blocks of a graph are pairwise edge disjoint and that the set of bridges forms a forest. For the set of blocks and the forest formed by the bridges of a graph G it holds that their cardinalities are bounded by $|V(G)|$ and they can be enumerated in time $O(|V(G)| + |E(G)|)$ [16].

Let G_1 and G_2 be graphs. G_1 and G_2 are *isomorphic*, denoted $G_1 \simeq G_2$, if there is a *bijection* $\varphi : V(G_1) \to V(G_2)$ such that (i) $\{u, v\} \in E(G_1)$ iff $\{\varphi(u), \varphi(v)\} \in E(G_2)$, (ii) $\lambda_{G_1}(u) = \lambda_{G_2}(\varphi(u))$, (iii) and if $\{u, v\} \in E(G_1)$ then $\lambda_{G_1}(\{u, v\}) = \lambda_{G_2}(\{\varphi(u), \varphi(v)\})$ hold for every $u, v \in V(G_1)$. In this paper, two graphs are considered to be the same if they are isomorphic. G_1 is *subgraph isomorphic* to G_2 if G_1 is isomorphic to a subgraph of G_2. Deciding whether a graph is subgraph isomorphic to another graph is NP-complete, as it generalizes e.g. the Hamiltonian path problem.

Outerplanar Graphs Informally, a graph is *planar* if it can be drawn in the plane in such a way that no two edges intersect except at a vertex in common. An *outerplanar graph* is a planar graph which can be embedded in the plane in such a way that all of its vertices lie on the boundary of the outer face. Throughout this work we consider connected outerplanar graphs and denote the set of connected outerplanar graphs over an alphabet Σ by $\mathcal{O}_\Sigma$. Clearly, trees are outerplanar graphs and hence, a graph is outerplanar iff each of its blocks is outerplanar [7]. Furthermore, as the blocks of a graph can be computed in linear time [16] and outerplanarity of a block can be decided also in linear time [10, 12], one can decide in linear time whether a graph is outerplanar.

A biconnected outerplanar graph G with n vertices contains at most $2n - 3$ edges and has a unique Hamiltonian cycle which bounds the outer face of a planar embedding of G [7]. This unique Hamiltonian cycle can be computed efficiently [10]. Thus, G can be considered as an n-polygon with at most $n - 3$ non-crossing diagonals. Below we state a bound for the number of cycles of G. Due to space limitation, we omit the proof.

Proposition 1 *A biconnected outerplanar graph with d diagonals has at most 2^{d+1} cycles.*

Given outerplanar graphs G and H, deciding whether H is subgraph isomorphic to G is an NP-complete problem. This follows from the fact that outerplanar graphs generalize forests and deciding whether a forest is subgraph isomorphic to a tree is NP-complete [6]. The following stronger negative result is shown in [15].

Theorem 2 *Deciding whether a connected outerplanar graph H is subgraph isomorphic to a biconnected outerplanar graph G is NP-complete.*

If, however, H is also biconnected, the following positive result holds [10].

Theorem 3 *Let G, H be biconnected outerplanar graphs. Then one can decide in time $O\left(|V(H)| \cdot |V(G)|^2\right)$ whether H is subgraph isomorphic to G.*

For the special case of trees, the following positive result holds [11].[5]

Theorem 4 *The problem whether a tree H is subgraph isomorphic to a tree G can be decided in time $O\left(|V(H)|^{1.5} \cdot |V(G)|\right)$.*

[5] The bound in Theorem 4 is improved by a log factor in [14]. For the sake of simplicity, we generalize the algorithm in [11] to outerplanar graphs in the long version of this paper. We note that the complexity of our algorithm can also be improved using the idea of [14].

3 The Problem Setting

In this section we define the frequent subgraph mining problem for a practically relevant class of outerplanar graphs with respect to a matching operator that preserves the pattern graph's bridge and block structure. To define the mining problem, we need the notions of tenuous outerplanar graphs and BBP subgraph isomorphism.

Tenuous Outerplanar Graphs Let $d \geq 0$ be some integer. A *d-tenuous* outerplanar graph G is an outerplanar graph such that each block of G has at most d diagonals. For an alphabet Σ and integer $d \geq 0$, $\mathcal{O}_\Sigma^d$ denotes the set of connected d-tenuous outerplanar graphs labeled by the elements of Σ. The class of d-tenuous outerplanar graphs forms a practically relevant graph class e.g. in chemoinformatics. As an example, out of the 250251 pharmacological molecules in the NCI dataset, 236180 (i.e., 94.3%) compounds have an outerplanar molecular graph. Furthermore, among the outerplanar compounds, there is no molecular graph having a block with more than 11 diagonals. In fact, there is only one compound containing a block with 11 diagonals; 236083 (i.e., 99.99%) compounds among the outerplanar graphs have at most 5 diagonals per block.

BBP Subgraph Isomorphism We continue our problem definition by introducing a matching operator between outerplanar graphs. Let $G, H \in \mathcal{O}_\Sigma$. A *bridge and block preserving* (BBP) subgraph isomorphism from H to G, denoted $H \preccurlyeq_{BBP} G$, is a subgraph isomorphism from H to G mapping (i) the set of bridges of H to the set of bridges of G and (ii) different blocks of H to different blocks of G. Notice that for trees, which are special outerplanar graphs (i.e., block-free), BBP subgraph isomorphism is equivalent to the ordinary subtree isomorphism. Thus, BBP subgraph isomorphism can be considered as a generalization of subtree isomorphism to outerplanar graphs which is more specific than ordinary subgraph isomorphism.

Besides complexity reasons raised by Theorem 2, the use of BBP subgraph isomorphism as matching operator is motivated by recent results in chemoinformatics which indicate that more powerful predictors can be obtained by considering matching operators that map certain fragments of the pattern molecule to certain fragments of the target molecule. One natural step towards this direction is to require that only ring structures (i.e., blocks) can be mapped to ring structures and that edge disjoint ring structures are mapped to edge disjoint ring structures.

The FTOSM Problem Using the above notions, we define the *frequent d-tenuous outerplanar subgraph mining problem* (FTOSM) as follows: *Given* (i) an alphabet Σ, (ii) a finite set $\mathcal{D} \subseteq \mathcal{O}_\Sigma^d$ of *transactions* for some integer $d \geq 0$, and (iii) an integer threshold $t > 0$, *enumerate* the set of all connected d-tenuous outerplanar graphs in $\mathcal{O}_\Sigma^d$ that match at least t graphs in $\mathcal{D}$ w.r.t. BBP subgraph isomorphism, i.e., enumerate the set

$$\mathcal{F}_{\Sigma,d}^t(\mathcal{D}) = \{H \in \mathcal{O}_\Sigma^d : \Pi_t(\mathcal{D}, H)\} \ , \tag{1}$$

where $\Pi_t(\mathcal{D}, H)$ is the *frequency property* defined by

$$\Pi_t(\mathcal{D}, H) = |\{G \in \mathcal{D} : H \preccurlyeq_{BBP} G\}| \geq t \ . \tag{2}$$

By definition, $\mathcal{F}^t_{\Sigma,d}(\mathcal{D})$ does not contain isomorphic graphs. Furthermore, it is closed downwards w.r.t. BBP subgraph isomorphism, i.e., $G_1 \in \mathcal{F}^t_{\Sigma,d}(\mathcal{D})$ whenever $G_2 \in \mathcal{F}^t_{\Sigma,d}(\mathcal{D})$ and $G_1 \preccurlyeq_{BBP} G_2$. Given $\mathcal{D}$ and t, we call a graph H satisfying (2) *t-frequent*.

The *parameters* of the FTOSM problem are the cardinality of the transaction dataset (i.e., $|\mathcal{D}|$) and the size of the largest graph in $\mathcal{D}$ (i.e., $\max\{|V(G)| : G \in \mathcal{D}\}$). Since d is usually small, it is assumed to be a *constant*. Note that the cardinality of $\mathcal{F}^t_{\Sigma,d}(\mathcal{D})$ can be exponential in the above parameters of $\mathcal{D}$. Clearly, in such cases it is impossible to enumerate $\mathcal{F}^t_{\Sigma,d}(\mathcal{D})$ in time polynomial in the parameters of $\mathcal{D}$. We therefore ask whether the FTOSM problem can be solved in *incremental polynomial time* (see, e.g., [9]), that is, whether there exists an enumeration algorithm listing the first k elements of $\mathcal{F}^t_{\Sigma,d}(\mathcal{D})$ in time polynomial in the *combined size* of $\mathcal{D}$ and the set of these k elements for every $k = 1, \ldots, |\mathcal{F}^t_{\Sigma,d}(\mathcal{D})|$.

We note that in the literature (see, e.g., [9]) one usually considers also the notion of *output polynomial time* (or *polynomial total time*) complexity for enumeration algorithms. Algorithms in this more liberal class are required to enumerate a set S in the combined size of the input and the *entire* set S. Thus, in contrast to incremental polynomial time, an output polynomial time algorithm may have in worst-case a delay time exponential in the size of the input before printing the kth element for some $k \geq 1$.

Although several algorithms mining frequent connected subgraphs from datasets of arbitrary graphs w.r.t. subgraph isomorphism have demonstrated their performance empirically, we note that this general problem cannot be solved in output polynomial time, unless P = NP. On the other hand, the frequent graph mining problem is solvable in incremental polynomial time when the graphs in the dataset are restricted to forests and the patterns to trees. This follows e.g. from the results in [3]. Since tenuous outerplanar graphs form a practically relevant graph class that naturally generalizes trees, by considering the FTOSM problem we take a step towards going beyond trees in frequent subgraph mining.

4 The Mining Algorithm

In this section we present Algorithm 1, an Apriori-like [1] algorithm, that solves the FTOSM problem in incremental polynomial time. For a set $\mathcal{D} \subseteq \mathcal{O}^d_\Sigma$ and integer $t \geq 0$, the algorithm computes iteratively the set of t-frequent k-patterns from the set of t-frequent $(k-1)$-patterns. A *k-pattern* is a graph $G \in \mathcal{O}^d_\Sigma$ such that the sum of the number of blocks of G and the number of vertices of G not belonging to any block is k.

In step 1 of the algorithm, we first compute the set of t-frequent 1-patterns, that is, the set of t-frequent graphs consisting of either a single vertex or a single block. The first set, denoted by $\mathcal{F}_v$ in step 1, can be computed in linear time. The second set, denoted $\mathcal{F}_b$, can be computed in time polynomial in the parameters of $\mathcal{D}$; an efficient Apriori-based algorithm for this problem is presented in Section 4.2.

In step 2 of the algorithm, we then compute the set of t-frequent 2-patterns, i.e., the set of graphs in $\mathcal{O}^d_\Sigma$ consisting of either (1) a single edge or (2) two blocks having a common vertex or (3) a block and a bridge edge having a common vertex. We denote the corresponding three sets in step 2 by $\mathcal{F}_e$, $\mathcal{F}_{bb}$, and $\mathcal{F}_{be}$, respectively. In the definitions of $\mathcal{F}_{bb}$ and $\mathcal{F}_{be}$, $G_1 \bowtie G_2$ denotes the set of graphs that can be obtained from the

Algorithm 1 FREQUENTOUTERPLANARGRAPHS

Require: $\mathcal{D} \subseteq \mathcal{O}_\Sigma^d$ for some alphabet Σ and integer $d \geq 0$, and integer $t > 0$
Ensure: $\mathcal{F}_{\Sigma,d}^t(\mathcal{D})$ defined in Eq. (1)

1: $\mathcal{L}_1 = \mathcal{F}_v \cup \mathcal{F}_b$, where

$$\mathcal{F}_v = \{H \in \mathcal{O}_\Sigma : |V(H)| = 1 \wedge \Pi_t(\mathcal{D}, H)\}$$
$$\mathcal{F}_b = \{H \in \mathcal{O}_\Sigma^d : H \text{ is biconnected} \wedge \Pi_t(\mathcal{D}, H)\}$$

2: $\mathcal{L}_2 = \mathcal{F}_e \cup \mathcal{F}_{bb} \cup \mathcal{F}_{be}$, where

$$\mathcal{F}_e = \{H \in \mathcal{O}_\Sigma : |E(H)| = 1 \wedge \Pi_t(\mathcal{D}, H)\}$$
$$\mathcal{F}_{bb} = \{H \in G_1 \bowtie G_2 : G_1, G_2 \in \mathcal{F}_b \wedge \Pi_t(\mathcal{D}, H)\}$$
$$\mathcal{F}_{be} = \{H \in G_1 \bowtie G_2 : G_1 \in \mathcal{F}_b \wedge G_2 \in \mathcal{F}_e \wedge \Pi_t(\mathcal{D}, H)\}$$

3: $k = 2$
4: **while** $\mathcal{L}_k \neq \emptyset$ **do**
5: $\quad k = k + 1$
6: $\quad \mathcal{C}_k = \text{GENERATECANDIDATES}(\mathcal{L}_{k-1})$
7: $\quad \mathcal{L}_k = \{H \in \mathcal{C}_k : \Pi_t(\mathcal{D}, H)\}$
8: **endwhile**
9: **return** $\cup_{i=1}^k \mathcal{L}_i$

union of G_1 and G_2 by contracting[6] a vertex from G_1 with a vertex from G_2 that have the same label. Clearly, $G_1 \bowtie G_2 \subseteq \mathcal{O}_\Sigma^d$ for every $G_1, G_2 \in \mathcal{O}_\Sigma^d$. The set $\mathcal{F}_e$ of t-frequent edges can be computed in linear time. Since the cardinalities of both $\mathcal{F}_{bb}$ and $\mathcal{F}_{be}$ are polynomial in the parameters of $\mathcal{D}$, and BBP subgraph isomorphism between outerplanar graphs can be decided in polynomial time by the result of Section 4.4 below, it follows that both $\mathcal{F}_{bb}$ and $\mathcal{F}_{be}$ and hence, the set $\mathcal{L}_2$ of t-frequent 2-patterns can be computed in time polynomial in the parameters of $\mathcal{D}$.

In the loop 4–8, we compute the set of t-frequent k-patterns for $k \geq 3$ in a way similar to the Apriori algorithm [1]. The crucial steps of the loop are the generation of candidate k-patterns from the set of t-frequent $(k-1)$-patterns (step 6) and the decision of t-frequency of the candidate patterns (step 7). In Sections 4.3 and 4.4 below we describe these steps in detail.

Putting together the results given in Theorems 7 – 10 stated in Sections 4.1 – 4.4, respectively, we can formulate the main result of this paper:

Theorem 5 *Algorithm 1 is correct and solves the FTOSM problem in incremental polynomial time.*

Before going into the technical details in Sections 4.1 – 4.4, we first describe a transformation on outerplanar graphs by means of block contraction that is used in different steps of the mining algorithm. More precisely, for a graph $G \in \mathcal{O}_\Sigma$, let $\widetilde{G}$ denote the

[6] The contraction of the vertices u and v of a graph G is the graph obtained from G by introducing a new vertex w, connecting w with every vertex in $N(u) \cup N(v)$, and removing u and v, as well as the edges adjacent to them.

graph over the alphabet $\Sigma \cup \{\#\}$ derived from G by the following transformation: For each block B in G, (i) introduce a new vertex v_B and label it by $\#$, (ii) remove each edge belonging to B, and (iii) for every vertex v of B, connect v with v_B by an edge labeled by $\#$, if v is adjacent to a bridge or to another block of G; otherwise remove v. In the following proposition we state some basic properties of $\widetilde{G}$.

Proposition 6 *Let $G \in \mathcal{O}_\Sigma$. Then*

(i) $|V(\widetilde{G})| = 1$ *iff* $|V(G)| = 1$ *or G is biconnected,*
(ii) for every $e \in E(\widetilde{G})$, at most one vertex of e is labeled by $\#$, and
(iii) $\widetilde{G}$ is a free tree.

Since $\widetilde{G}$ is a tree, we call it the *block and bridge tree (BB-tree)* of G.

4.1 Canonical String Representation

One time consuming step of mining frequent d-tenuous outerplanar graphs is to test whether a particular graph $H \in \mathcal{O}_\Sigma^d$ belongs to some subset S of $\mathcal{O}_\Sigma^d$. To apply advanced data structures that allow fast search in large subsets of $\mathcal{O}_\Sigma^d$, we need to define a total order on $\mathcal{O}_\Sigma^d$. Similarly to many other frequent graph mining algorithms, we solve this problem by assigning a *canonical string* to each element of $\mathcal{O}_\Sigma$ such that (i) two graphs have the same canonical string iff they are isomorphic and (ii) for every $G \in \mathcal{O}_\Sigma$, the canonical string of G can be computed efficiently. Using some canonical string representation satisfying the above properties, a total order on $\mathcal{O}_\Sigma$ and thus, on $\mathcal{O}_\Sigma^d$ as well, can be defined by some total order (e.g. lexicographic) on the set of strings assigned to the elements of $\mathcal{O}_\Sigma$. Furthermore, property (i) allows one to decide isomorphism between two outerplanar graphs by comparing their canonical strings.

Although the canonical string representation for outerplanar graphs may be of some interest in itself, due to space limitations we omit its definition which is based on the BB-tree $\widetilde{G}$ of G. By (iii) of Proposition 6, $\widetilde{G}$ is a free tree. Utilizing this property, we can generalize the depth-first canonical representation for free trees (see, e.g., [2]) to outerplanar graphs, and state the following result:

Theorem 7 *A canonical string representation of a graph in $\mathcal{O}_\Sigma$ with n vertices can be computed in time $O\left(n^2 \log n\right)$.*

4.2 Mining Frequent Biconnected Graphs

In this section we present Algorithm 2, an Apriori-like algorithm, that computes the set $\mathcal{F}_b$ of t-frequent d-tenuous biconnected graphs used in step 1 of Algorithm 1. Since d is constant, Algorithm 2 runs in time polynomial in the parameters of $\mathcal{D}$.

In step 1 of Algorithm 2, we first compute the set $\mathcal{L}_0$ of t-frequent cycles as follows: We list the cycles of G for every $G \in \mathcal{D}$ and count their frequencies. Proposition 1 in Section 2 implies that the number of cycles of a d-tenuous outerplanar graph G is bounded by $O\left(|V(G)|\right)$ if d is assumed to be constant. Furthermore, from [13, 16] it follows that the cycles of a graph can be listed with linear delay. Since isomorphism

Algorithm 2 FREQUENTBICONNECTEDGRAPHS

Require: $\mathcal{D} \subseteq \mathcal{O}_\Sigma^d$ for some alphabet Σ and integer $d \geq 0$, and integer $t > 0$
Ensure: $\mathcal{F}_b$ defined in step 1 of Algorithm 1

1: **let** $\mathcal{L}_0 \subseteq \mathcal{O}_\Sigma^0$ be the set of t-frequent cycles in $\mathcal{D}$
2: **for** $k = 1$ **to** d **do**
3: **let** $\mathcal{C}_k \subseteq \mathcal{O}_\Sigma^k \setminus \mathcal{O}_\Sigma^{k-1}$ be the set of biconnected graphs H such that $H \ominus \Delta \in \mathcal{L}_{k-1}$ for every diagonal Δ of H
4: $\mathcal{L}_k = \{H \in \mathcal{C}_k : \Pi_t(\mathcal{D}, H)\}$
5: **endfor**
6: **return** $\bigcup_{k=0}^{d} \mathcal{L}_k$

between cycles can be decided efficiently, these results together imply that $\mathcal{L}_0$ can be computed in time polynomial in the parameters of $\mathcal{D}$.

In loop 2–5 of Algorithm 2, we compute the sets of t-frequent biconnected graphs containing k diagonals for every $k = 1, \ldots, d$. In particular, in step 3 we compute the set $\mathcal{C}_k$ of candidate biconnected graphs $H \in \mathcal{O}_\Sigma^k$ satisfying the following conditions: H has exactly k diagonals and the removal of any diagonal from H, denoted by $\ominus$ in step 3, results in a t-frequent biconnected graph. Putting the above results together, we can state the following theorem. (We omit the proof in this short version.)

Theorem 8 *Algorithm 2 is correct and computes the set of t-frequent d-tenuous biconnected outerplanar graphs in time polynomial in the parameters of $\mathcal{D}$.*

4.3 Candidate Generation

In step 6 of Algorithm 1, we generate the set of candidate k-patterns. In this section we give Algorithm 3, a generalization of the candidate generation algorithm for free trees described in [3], that computes the set of candidate k-patterns from the set of frequent $(k-1)$-patterns. Applying the candidate generation principle of the Apriori algorithm [1], each candidate is obtained by joining two frequent $(k-1)$-patterns that have an isomorphic $(k-2)$-pattern core.

In the outer loop 2–12 of the algorithm, we consider each possible pair G_1, G_2 of frequent $(k-1)$-patterns, and in loop 3–11, each pair g_1 and g_2 of *leaf* subgraphs of G_1 and G_2, respectively. By a leaf subgraph of a k-pattern H for $k \geq 2$ we mean the subgraph of H represented by a leaf of the BB-tree $\widetilde{H}$. If G_1 and G_2 are the same graphs then, for completeness, we consider also the case when g_1 and g_2 are isomorphic leaf subgraphs. We remove g_1 and g_2 from G_1 and G_2, respectively, denoted by $\ominus$ in the algorithm, and check whether the obtained graphs G_1' and G_2' are isomorphic (step 4). The removal of a biconnected component means the deletion of each of its edges and vertices except the distinguished vertex which is adjacent to a bridge or to another block.

If G_1' and G_2' are isomorphic then we consider every leaf subgraph g_1' of G_1' (loop 5–10) and check whether g_2 can be attached to g_1' in G_1 consistently with G_2 (step 6). More precisely, let g_2' be a block or a vertex not belonging to a block in G_2 such that g_2 is hanging from g_2', i.e., the only edge adjacent to g_2 is adjacent also to g_2'. We say that g_2 can be attached to g_1' in G_1 consistently with G_2 if g_1' is isomorphic to g_2'. Thus, if

Algorithm 3 GENERATECANDIDATES

Require: set $\mathcal{L}_{k-1}$ of frequent $k-1$-patterns for some $k > 2$
Ensure: set $\mathcal{C}_k$ of candidate k-patterns

1: $\mathcal{C}_k = \emptyset$
2: **forall** $G_1, G_2 \in \mathcal{L}_{k-1}$ **do**
3: **forall** $g_1 \in \text{Leaf}(G_1)$ and $g_2 \in \text{Leaf}(G_2)$ **do**
4: **if** $G_1 \ominus g_1 \simeq G_2 \ominus g_2$ **then**
5: **forall** $g_1' \in \text{Leaf}(G_1 \ominus g_1)$ **do**
6: **if** g_2 is attachable to g_1' consistently with G_2 **then**
7: attach g_2 in G_1 to g_1' consistently with G_2 and denote the obtained graph by C
8: **if** g_1, g_2 have the top two string encodings in C, $C \notin C_k$, and
$C \ominus g \in \mathcal{L}_{k-1}$ for every $g \in \text{Leaf}(C)$
9: **then** add C to $\mathcal{C}_k$
10: **endfor**
11: **endfor**
12: **endfor**
13: **return** $\mathcal{C}_k$

the condition in step 6 holds then we attach g_2 to g_1' consistently with G_2 and denote the obtained graph by C (step 7).

Notice that C can be generated in many different ways, depending on the particular choice of g_1 and g_2. To reduce the amount of unnecessary computation, we consider only those pairs which are among the top leaf subgraphs of C, i.e., which have the top two string encodings w.r.t. a center of $\tilde{C}$. By definition, a vertex representing a leaf subgraph of C is always a leaf in $\tilde{C}$. If this condition holds then we add C to the set of candidates in step 9 if for every leaf subgraph g of C, the $(k-1)$-pattern obtained from C by removing g is frequent (see step 8). We omit the proof of the following theorem.

Theorem 9 *Let $\mathcal{C}_k$ be the output of Algorithm 3 and $\mathcal{L}_k$ the set of frequent k-patterns for any $k > 2$. Then $\mathcal{L}_k \subseteq \mathcal{C}_k$, the cardinality of $\mathcal{C}_k$ is polynomial in the cardinality of $\mathcal{L}_{k-1}$, and $\mathcal{C}_k$ can be computed in time polynomial in the size of $\mathcal{L}_{k-1}$.*

4.4 BBP Subgraph Isomorphism

Algorithms 1 and 2 contain the steps of deciding whether a candidate pattern $H \in \mathcal{O}_\Sigma^d$ is t-frequent, i.e., whether it is BBP subgraph isomorphic to at least t graphs in $\mathcal{D}$. While subgraph isomorphism between outerplanar graphs is NP-complete even for very restricted cases (see Theorem 2), Theorem 10, the main result of this section, states that BBP subgraph isomorphism can be decided efficiently between outerplanar graphs if the pattern graph H is connected. The connectivity is necessary, as otherwise the problem would generalize the NP-complete subforest isomorphism problem [6]. We note that the result of Theorem 10 generalizes the positive result on subtree isomorphism given in Theorem 4 and may thus be of some interest in itself.

Theorem 10 *Let $G, H \in \mathcal{O}_\Sigma$ such that H is connected. Then $H \preccurlyeq_{BBP} G$ can be decided in polynomial time.*

Table 1. Number of patterns (#C), number of frequent patterns (#FP), and runtime in seconds for candidate generation and evaluation (T) with frequency thresholds 10%, 5%, 2%, and 1%

size	10%			5%			2%			1%		
(k)	#C	#FP	T	#C	#FP	T	#C	#FP	T	#C	#FP	T
1	86	7	107	144	11	169	582	25	380	2196	55	824
2	74	16	446	216	24	570	1332	61	1118	6208	174	2554
3	139	41	1133	234	74	1393	510	170	2123	1516	659	5653
4	133	77	1232	266	154	2038	642	356	4079	2554	1776	11899
5	139	91	1071	319	222	2268	909	644	5603	4550	3886	20411
6	107	72	754	332	252	1847	1212	918	6105	7314	6490	28811
7	61	41	472	295	195	1168	1266	990	4964	10165	9058	34967
8	37	25	354	182	137	741	1086	893	3384	11479	10396	36391
9	20	13	205	137	116	602	956	803	2282	11129	10194	31721
10	8	5	130	131	119	594	828	700	1635	9370	8623	23412
11	0	0	0	131	117	565	697	604	1360	7276	6818	15530
12	0	0	0	115	107	536	707	665	1483	5533	5184	9345
13	0	0	0	78	64	412	1027	1022	2017	4395	4145	5252
14	0	0	0	27	21	250	1702	1700	2858	4303	4194	3707
15	0	0	0	4	3	89	2725	2715	3957	5422	5376	4089

Due to space limitations, we omit the proof of the above theorem. We only note that the algorithm first computes the BB-trees of the input graphs G and H, and then combines the subgraph isomorphism algorithms between labeled trees (generalization of [11]) and labeled biconnected outerplanar graphs (generalization of [10]).

5 Experimental Evaluation

In our experiments, we used the NCI dataset consisting of 250251 chemical compounds. For our work, it was important to recognize that 236180 (i.e., 94.3%) of these compounds have outerplanar molecular graph. Thus, outerplanar graphs form a practically relevant class of graphs. Among the outerplanar molecular graphs, there are 21963 trees (i.e., 8.8% of the outerplanar subset). In the experiments, we have removed the non-outerplanar graphs from the dataset. Altogether, the outerplanar molecules contain 423378 blocks, with up to 11 diagonals per block. However, 236083 (i.e., 99.99%) of the outerplanar molecular graphs have at most 5 diagonals per block. This empirical observation validates our approach to assume the number of diagonals to be constant.

The database contains a wide variety of structures, and a low relative frequency threshold is needed to mine a significant number of patterns. E.g. though there are 15426 pairwise non-isomorphic cycles in the database, only a few of them are really frequent; the only one above 10% is the benzene ring with frequency 66%.

Our results are given in Table 1. It shows the number of candidate (#C) and frequent (#FP) k-patterns discovered for $k = 1, \ldots, 15$, as well as the runtime (T) in seconds for the computation and evaluation of the candidates using the frequency thresholds 10%, 5%, 2% and 1%. As expected, the number and the size of the discovered patterns is much larger when the frequency threshold is lower. Even though the embeddings of

$(k-1)$ patterns are computed (again) in level k, the time needed to complete one level does not necessarily increase with k. It is interesting to note that after the number of frequent k-patterns drops a bit when k gets larger then 8, this number again increases when k exceeds 12, and the number of frequent patterns gets close to the number of candidate patterns. This is because this particular dataset contains large subsets with molecules sharing large biconnected structures (such as the HIV active substance dataset). The time needed for candidate generation is always smaller than 1% of the total time. The time needed for coverage testing per pattern depends on how much structure these patterns share. If the number of patterns is large, the time needed per pattern is usually lower.

One can make several conclusions. First, our algorithm can mine an expressive class of molecular patterns from a relatively large database. Although the presented experiments happened entirely in memory (taking about 600Mb), our approach does not depend on storing intermediate results in memory between the different passes over the database. This means that we could also perform this algorithm with a database on disk. In our application e.g., this would bring an overhead of about 15 seconds per pass over the database. Second, we can conclude that the complexity of the coverage testing scales well as the pattern size grows, as predicted by theory. In this application, due to the implementation exploiting shared structure among patterns, the time needed for evaluation per pattern does not even depend in a clear systematic way on the pattern size.

6 Conclusion and Open Problems

We have defined the FTOSM problem motivated by chemical datasets and presented an Apriori-based algorithm solving this enumeration problem in incremental-polynomial time. To the best of our knowledge, no fragment of the frequent subgraph mining problem *beyond trees* has so far been identified, for which the problem can be solved in incremental polynomial time. Our algorithm is based on a canonical string representation of outerplanar graphs and further algorithmic components for mining frequent biconnected outerplanar graphs and candidate generation in an Apriori style algorithm. Motivated by application and complexity considerations, we introduced a special kind of subgraph isomorphism which generalizes subtree isomorphism but is at the same time more specific than ordinary subgraph isomorphism, and which is decidable in polynomial time for outerplanar graphs. We presented also empirical results with a large dataset indicating the effective practical performance of our algorithm. We believe that the identification of tractable practical fragments of the frequent subgraph mining problem is an important challenge for the data mining community.

Besides working on optimization of the algorithm, e.g., on improving the time complexity of the coverage testing, it is natural to ask whether the positive result of this paper can be generalized to arbitrary outerplanar graphs. Notice that our algorithm exploits the constant bound on the number of diagonals only in the computation of the set $\mathcal{F}_b$ of frequent biconnected graphs in step 1 of Algorithm 1. Therefore, to generalize the result of this paper to arbitrary outerplanar graphs, it is sufficient to consider the following special problem: *Given* a finite set $\mathcal{D} \subseteq \mathcal{O}_\Sigma$ of biconnected outerplanar graphs and a

non-negative integer t, compute the set of t-frequent patterns in $\mathcal{D}$ w.r.t. BBP subgraph isomorphism. Notice that this problem definition implicitly requires t-frequent patterns to be biconnected because by definition, there is no BBP subgraph isomorphism from a non-biconnected graph to a biconnected outerplanar graph. We do not know whether this special problem can be solved in incremental or at least in output polynomial time.

Acknowledgments

Tamás Horváth and Stefan Wrobel were partially supported by the DFG project (WR 40/2-1) *Hybride Methoden und Systemarchitekturen für heterogene Informationsräume*. Jan Ramon is a post-doctoral fellow of the Fund for Scientific Research (FWO) of Flanders.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pp. 307–328. AAAI/MIT Press, 1996.
2. Y. Chi, R. R. Muntz, S. Nijssen, and J. N. Kok. Frequent subtree mining - An overview. *Fundamenta Informaticae*, 66(1-2):161–198, 2005.
3. Y. Chi, Y. Yang, and R. R. Muntz. Canonical forms for labelled trees and their applications in frequent subtree mining. *Knowledge and Information Systems*, 8(2):203–234, 2005.
4. D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *J. of Artificial Intelligence Research*, 1:231–255, 1994.
5. M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1036–1050, 2005.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to NP-Completeness*. Freeman, San Francisco, CA, 1979.
7. F. Harary. *Graph Theory*. Addison–Wesley, Reading, Massachusetts, 1971.
8. A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354, 2003.
9. D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.
10. A. Lingas. Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoretical Computer Science*, 63:295–302, 1989.
11. D. W. Matula. Subtree isomorphism in $O(n^{\frac{5}{2}})$. *Annals of Discrete Mathematics*, 2:91–106, 1978.
12. S. L. Mitchell. Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Information Processing Letters*, 9(5):229–232, 1979.
13. R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.
14. R. Shamir and D. Tsur. Faster subtree isomorphism. *J. of Algorithms*, 33(2):267–280, 1999.
15. M. M. Sysło. The subgraph isomorphism problem for outerplanar graphs. *Theoretical Computer Science*, 17:91–97, 1982.
16. R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. on Computing*, 1(2):146–160, 1972.
17. X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proc. of the 2002 IEEE Int. Conference on Data Mining (ICDM)*, pp. 721–724, IEEE Computer Society, 2002.

Type Extension Trees: a Unified Framework for Relational Feature Construction

Manfred Jaeger

Institut for Datalogi, Aalborg Universitet, Fredrik Bajers Vej 7E, DK-9220 Aalborg Ø
jaeger@cs.aau.dk

Abstract. We introduce type extension trees as a formal representation language for complex combinatorial features of relational data. Based on a very simple syntax this language provides a unified framework for expressing features as diverse as embedded subgraphs on the one hand, and marginal counts of attribute values on the other. We show by various examples how many existing relational data mining techniques can be expressed as the problem of constructing a type extension tree and a discriminant function.

1 Introduction

A key component of relational data mining methods is the construction of relevant features. Whereas in conventional ("propositional") learning settings the set of possible features is usually given by the available attributes, one has in relational learning the ability to construct new features by considering the relational neighborhood of an entity. Taking into consideration related entities and their attributes, one obtains a basically unlimited supply of potential features. The space of features actually explored by specific relational learning methods is often not very clearly defined and more or less only implicit in the learning algorithm and its candidate feature generation mechanisms.

In this paper we study relational features in their own right. We propose type extension trees (TETs) as a very simple, yet general and powerful representation language for relational features. We will illustrate the expressiveness and flexibility of type extension trees by showing how they can represent a great variety of different types of features used by different kinds of methods in graph mining, relational learning, and standard propositional learning.

This paper is mostly conceptual. Though intended to be eventually used in an implemented learning system, we introduce type extension trees in this paper mostly as a basis for an integrated view of existing data mining models and techniques. TETs provide a unified view along several dimensions: they accommodate relational features as considered in a variety of different disciplines, ranging from graph mining to statistical relational learning and probabilistic inductive logic programming (ILP); they provide a unified view of features of single entities, tuples of entities, or whole datasets; they support all levels of separation or integration of the feature construction and model induction components in a relational learning procedure.

Type extension trees, thus, provide a common ground from which fundamental aspects of different data mining techniques can be more clearly elucidated. A unifying conceptual framework is all but indispensable for a theoretical analysis of differences and similarities between different data mining techniques, and can help to translate results and techniques across different areas.

2 Type Extension Trees

In this section we introduce syntax and semantics of type extension trees. Our definitions are partly motivated by *type extension axioms*, which play a pivotal role in finite model theory [4]. Type extension axioms and various generalizations have been used to characterize the combinatorial structure of large random structures [5, 13, 8, 1]. In the finite model theory setting the generating distribution is given, and one is interested in the asymptotic properties of random structures. In relational learning the situation is reversed: one observes one randomly generated structure (the data), and would like to infer a model for the generating process. A language that has been found to express the characteristic features of data sampled from known distributions is also a good candidate for specifying those features of data that are important for reconstructing an unknown generating model.

We now turn to defining type extension trees, starting with definitions relating to relational structures, which serve as a general, abstract model for structured data.

R denotes a *relational signature* (or vocabulary), i.e. a set of relation symbols (of any arities). We typically use $r, s, t, \ldots$ to denote symbols from R, and $|\,r\,|$ to denote the arity of r. Throughout, we denote with u, v, w logical variables, and with $a, b, c, \ldots$ constants (representing specific entities in a domain). Tuples of variables and constants are denoted in bold font: $\boldsymbol{v}, \boldsymbol{a}$, etc. The length of a tuple $\boldsymbol{v}$ is denoted $|\,\boldsymbol{v}\,|$. An expression of the form $r(u, v), r(a, v), etc.$ is called an *atom*. If all arguments are constants, the atom is *ground*. A *literal* is an atom or a negated atom. A *type* is a conjunction of literals. Types are denoted $\tau, \sigma, \ldots$. We use T to denote an empty conjunction, which is to be interpreted as a tautology.

Definition 1. *A* relational R-structure $\mathcal{M}$ *consists of a domain* M *and an* interpretation function $I^{\mathcal{M}}$ *that assigns truth values to ground* R*-atoms, i.e.* $I^{\mathcal{M}}(r(\boldsymbol{a})) \in \{\text{true}, \text{false}\}$ *for all* $r \in R$ *and* $\boldsymbol{a} \in M^{|r|}$. *The* size *of* $\mathcal{M}$ *is the cardinality of* M *(in this paper always assumed to be finite).*

In logic programming terminology, an R-structure is just an interpretation. An R-structure with only unary and binary relations can be seen as a graph with colored nodes and edges. From a database perspective, an R-structure is a relational database with only boolean attributes. Using a boolean encoding of categorical attributes, one can represent databases with only categorical attributes as relational structures in the sense of definition 1.

Definition 2. *A* type extension tree (TET) *over R is a tree whose nodes are labeled with R-types, and whose edges are labeled with (possibly empty) sets of variables.*

A type extension tree is syntactically closely related to predicate logic formulas, with subtrees corresponding to sub-formulas, and edge labels corresponding to variables that are quantified over. In analogy to standard predicate logic definitions one defines the *free variables* of a TET, and the *substitution* of constants $\boldsymbol{a}$ for free variables $\boldsymbol{v}$.

TETs can be represented graphically as in figure 4 or equation (1). We also write $[\tau, [\boldsymbol{w}_1, T_1], \ldots, [\boldsymbol{w}_m, T_m]]$ for a TET whose root is labeled with τ, and which has m subtrees $T_1, \ldots, T_m$ reached by edges with labels $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m$. We write $T(\boldsymbol{v})$ for a TET containing free variables $\boldsymbol{v}$, and $T(\boldsymbol{a})$ for the result of substituting $\boldsymbol{a}$ for $\boldsymbol{v}$ in T.

The semantics of TETs differs from semantics of most formal languages in that interpretations of TETs do not just return a truth value, but a complex combinatorial structure. To motivate the following definitions, consider the first-order sentence $\tau(\boldsymbol{a}) \rightarrow \exists w \sigma(\boldsymbol{a}, w)$. Interpreted over a structure $\mathcal{M}$ with $\boldsymbol{a} \subseteq M$ this sentence is either true or false. Furthermore, one can replace the existential quantifier $\exists$ with a counting quantifier like $\exists^{\geq 10}$ stipulating the existence of at least ten w with $\sigma(\boldsymbol{a}, w)$. The interpretation of the new sentence would still be either true or false. However, instead of specifying exact truth conditions for the quantifier in front of w, we can also define the interpretation of the formula as the number of $b \in M$ such that $\sigma(\boldsymbol{a}, b)$ is true. This is basically how we will interpret the TET $\tau(\boldsymbol{a}) \overset{w}{-} \sigma(\boldsymbol{a}, w)$. For more complex TETs the interpretations are not simply numbers, but more complex combinatorial specifications, which are introduced by the following two definitions.

Definition 3. *For a finite set D and $k \in \mathbb{N}$ define the set of all k-multisets over D as*

$$\text{multisets}(D, k) := \{f : D \rightarrow \{0, \ldots, k\} \mid \sum_{d \in D} f(d) = k\}$$

Concrete multisets are written in the notation $[d_1 \mapsto k_1, \ldots, d_q \mapsto k_q]$ (only listing the $d_i \in D$ with $f(d_i) = k_i > 0$).

Definition 4. *The* value space $\mathcal{V}_n(T)$ *of a TET T for structures of size n is inductively defined as follows:*

- *If T consists of a single node, then $\mathcal{V}_n(T) = \{\text{true}, \text{false}\}$.*
- *If $T = [\tau, [\boldsymbol{w}_1, T_1], \ldots, [\boldsymbol{w}_m, T_m]]$, then*

$$\mathcal{V}_n(T) = \{\text{true}, \text{false}\} \times \times_{i=1}^{m} \text{multisets}(\mathcal{V}_n(T_i), n^{|\boldsymbol{w}_i|})$$

Example 1. The value space of the TET $T(\boldsymbol{v}) = \tau(\boldsymbol{v}) \overset{w}{-} \sigma(\boldsymbol{v}, w)$ over a structure $\mathcal{M}$ of size n is $\mathcal{V}_n(T(\boldsymbol{v})) = \{\mathit{true}, \mathit{false}\} \times \mathit{multisets}(\{\mathit{true}, \mathit{false}\}, n)$. A tuple $\boldsymbol{a} \in \mathcal{M}^{|\boldsymbol{v}|}$ defines a value $V(T(\boldsymbol{a})) = (I, f) \in \mathcal{V}_n(T(\boldsymbol{v}))$, where $I \in \{\mathit{true}, \mathit{false}\}$ is the truth value of $\tau(\boldsymbol{a})$ in $\mathcal{M}$, and $f \in \mathit{multisets}(\{\mathit{true}, \mathit{false}\}, n)$ gives the counts

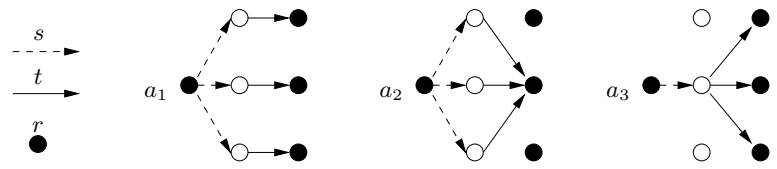


Fig. 1. Example 2

of elements $b \in M$ for which $\sigma(\boldsymbol{a}, b)$ is true, respectively false. The general definition of the value of a tuple $\boldsymbol{a}$ in a structure $\mathcal{M}$ is given in the following definition.

Definition 5. *Let $T(\boldsymbol{v})$ be a TET, $\mathcal{M}$ an R-structure of size n, and $\boldsymbol{a} \in M^{|\boldsymbol{v}|}$. The value of the ground TET $T(\boldsymbol{a})$ is defined as follows:*

- $V([\tau(\boldsymbol{a})]) = I^{\mathcal{M}}(\tau(\boldsymbol{a}))$
- $V([\tau(\boldsymbol{a}), [\boldsymbol{w}_1, T_1(\boldsymbol{a}, \boldsymbol{w}_1)], \ldots, [\boldsymbol{w}_m, T_m(\boldsymbol{a}, \boldsymbol{w}_m)]]) =$
  $(I^{\mathcal{M}}(\tau(\boldsymbol{a})), f(\boldsymbol{a}, \boldsymbol{w}_1, T_1), \ldots, f(\boldsymbol{a}, \boldsymbol{w}_m, T_m))$, *where*

$$f(\boldsymbol{a}, \boldsymbol{w}_i, T_i) : \gamma \mapsto |\{\boldsymbol{b} \in M^{|\boldsymbol{w}_i|} \mid V(T_i(\boldsymbol{a}, \boldsymbol{b})) = \gamma\}| \qquad (\gamma \in \mathcal{V}_n(T_i))$$

Example 2. Figure 1 shows three different structures for a vocabulary containing two binary relations s, t, and one unary relation r. The relational neighborhoods of the entities a_1, a_2, a_3 in the three structures have somewhat similar, yet unique, properties. Consider the following three TETs:

$$T_1(v) : \top(v) \overset{u,w}{-} s(v,u), t(u,w), r(w)$$
$$T_2(v) : \top(v) \overset{u}{-} s(v,u) \overset{w}{-} t(u,w), r(w)$$
$$T_3(v) : \top(v) \overset{w}{-} r(w) \overset{u}{-} s(v,u)t(u,w)$$

The value space $\mathcal{V}_7(T_1(v))$ is $\{t, f\} \times$ *multisets*$(\{t, f\}, 7^2)$. If M is a domain of seven entities, and $a \in M$, then $V(T_1(a)) = (t, [t \mapsto l, f \mapsto 7^2 - l])$, where l is the number of distinct tuples $(b, c) \in M^2$ for which $s(a, b) \wedge t(b, c) \wedge r(c)$ is true. Thus, $T_1(a)$ just gives the counts of paths leading from a via one $s()$ and one $t()$ relation to an entity with attribute $r()$. This count is 3 for a_1, a_2, a_3, so that $V(T_1(a_1)) = V(T_1(a_2)) = V(T_1(a_3))$.

The value space $\mathcal{V}_n(T_2)$ is more complex. Values here not only encode the number of $s - t$-paths to an entity with attribute r, but also differentiate with regard to the number of different intermediate nodes on these paths. The precise values are (for legibility, multisets here written as column vectors):

$$V(T_2(a_1)) = V(T_2(a_2)) = \left(t, \begin{bmatrix} \left(t, \begin{bmatrix} t \mapsto 1 \\ f \mapsto 6 \end{bmatrix}\right) \mapsto 3 \\ \left(f, \begin{bmatrix} f \mapsto 7 \end{bmatrix}\right) \mapsto 4 \end{bmatrix}\right) \qquad V(T_2(a_3)) = \left(t, \begin{bmatrix} \left(t, \begin{bmatrix} t \mapsto 3 \\ f \mapsto 4 \end{bmatrix}\right) \mapsto 1 \\ \left(f, \begin{bmatrix} f \mapsto 7 \end{bmatrix}\right) \mapsto 6 \end{bmatrix}\right)$$

Thus, T_2 distinguishes a_1 from a_3, but not from a_2. Another variation is provided by $T_3(v)$, which counts the number of distinct endpoints c (but not the number of distinct mid-points b): $V(T_3(a_1)) = V(T_3(a_3)) \neq V(T_3(a_2))$.

In the preceding example we found that type extension trees T_2, T_3 provided somewhat more information than T_1. This is formally captured in the following definition, which provides a refinement concept related to those used in ILP-based learning methods.

Definition 6. *Let $T(\boldsymbol{v}), T'(\boldsymbol{v})$ be type extension trees. T' is called a* refinement *of T if there exist functions*

$$\Gamma_n : \ \mathcal{V}_n(T') \to \mathcal{V}_n(T) \quad (n \in \mathbb{N}),$$

such that for all structures $\mathcal{M}$ of size n, and all $\boldsymbol{a} \in M^{|\boldsymbol{v}|}$:

$$V(T'(\boldsymbol{a})) = \gamma \to V(T(\boldsymbol{a})) = \Gamma_n(\gamma).$$

Here we have defined refinement on a semantic basis. One can easily define several syntactic operations on TETs that produce refinements (e.g. adding a new subtree to an existing TET).

Values $V(T(\boldsymbol{a}))$ provide a very detailed quantitative picture of the "relational neighborhood" of tuple $\boldsymbol{a}$ in structure $\mathcal{M}$. For all but the simplest TETs, the full value will be too complex to handle by a model induction algorithm. The complex feature value $V(T(\boldsymbol{a}))$, therefore, may need to be reduced or summarized.

Definition 7. *A* discriminant function *for a TET T is a function*

$$d : \cup_{n \geq 1} \mathcal{V}_n(T) \to \mathbb{R}.$$

Discriminant functions can be employed in different ways. For example, a 0,1-valued discriminant function turns T into a boolean feature. A collection of TETs, each equipped with a boolean discriminant function, then defines a set of boolean features that can be used by standard propositional model induction algorithms. However, discriminant functions can also represent the final model itself. For example, a discriminant function on $T(v)$ with values in $\{1, 2, 3\}$ could represent a predictive model for a three-state class variable $c(v)$.

3 Examples

In this section we show how several quite distinct relational data mining tasks can be represented as the problem of finding a TET and a discriminant function.

3.1 Frequent subgraphs

A key task in graph-based data mining is finding frequent subgraphs (see e.g. [19]). Consider, for example, a relational alphabet $R = \{r(\cdot), t(\cdot,\cdot)\}$. Figure 2 shows a two node graph G over this vocabulary.

Fig. 2. A two node subgraph

Now consider the TET

$$T_G(v,w) = r(v) \overset{\emptyset}{-} \neg r(w) \overset{\emptyset}{-} \neg t(v,v) \overset{\emptyset}{-} t(w,w) \overset{\emptyset}{-} t(v,w) \overset{\emptyset}{-} \neg t(w,v) \qquad (1)$$

The value space $\mathcal{V}_n(T_G(v,w))$ is equivalent to the 6-fold product of $\{true, false\}$ (independent of n). For any two entities a, b, the value $V(T_G(a,b))$ determines the truth value of all the ground atoms $r(a), r(b), \ldots, t(b,a)$, and thus represents the subgraph induced by a, b.

Now let d_e be the discriminant function on $\mathcal{V}(T_G)$ that counts the number of *false* components in $V(T_G)$. For any a, b, then $d_e(V(T_G(a,b)))$ is the edit distance between the subgraph induced by a, b and G.

Suppose, now, we want to determine whether our data $\mathcal{M}$ contains at least k occurrences of 2-node subgraphs whose edit distance to G is at most l. This becomes an evaluation of a TET-discriminant function by defining on the TET

$$T'_G = \top \overset{v,w}{-} T_G(v,w) \qquad (2)$$

the discriminant function $d_{k,l}$ defined by

$$d_{k,l}(V(\top), f((v,w), T_G(v,w)) = \begin{cases} 1 \text{ if } \sum_{\gamma \in \mathcal{V}(T_G(v,w)): d_e(\gamma) \leq l} f((v,w), T_G(v,w))(\gamma) & > k \\ 0 \text{ else} \end{cases}$$

Finally, the problem of finding all subgraphs G' that are "approximately frequent" (in the sense that graphs with edit distance to G' at most l occur at least k times) now becomes the problem of finding all TETs $T_{G'}$ of the structure $\top \overset{\boldsymbol{v}}{-} T_{G'}(\boldsymbol{v})$ with $d_{k,l}(V(T_{G'})) = 1$.

This example shows how TETs and discriminant functions capture operations at all stages of feature construction and model induction: a TET of the form (1) together with the discriminant function d_e is just a boolean feature that could be used by any model induction algorithm. However, full model induction tasks can also be expressed as a search for TETs and discriminant functions.

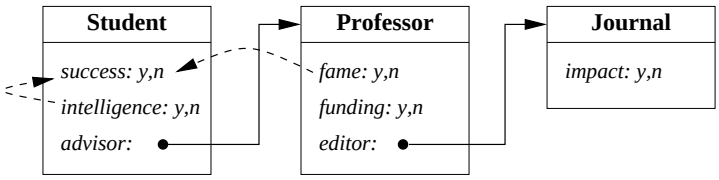


Fig. 3. A simple PRM

3.2 Probabilistic Relational Models

In this section we consider probabilistic relational models (PRMs) in the sense of [17, 6]. PRMs in their simplest form (there exist various extensions) are a probabilistic model for attribute values in a relational database. Figure 3 illustrates a simple example PRM based on [17]. It consists of a database schema consisting of three tables 'student', 'professor' and 'journal'. All tables have one or two boolean attributes. The student table also has a *reference attribute advisor*, which points to entries in the professor table, and the professor table has a reference attribute *editor* pointing to the journal table. The dashed arrows in figure 3 indicates that the PRM provides a probabilistic model for the *success* attribute in the student table, and that according to this model the probability for student.*success* values depends on the values of the *intelligence* attribute for the given student, and on the value of the *fame* attribute of the student's advisor. The complete specification of the PRM furthermore will contain the quantitative specification of the conditional distribution for student.*success*, which is not shown here.

Generally, the probabilities for attributes can be defined conditional on attribute values at any other table entries that can be reached by following a sequence of references. References can also be followed in inverse order: for example, by first following the *advisor* reference from a student entry s, one finds an entry p in the professor table; by then following the inverse of the *advisor* reference from p, one finds all entries in the student table that have the same advisor as s. Similarly, the student.*success* attribute could also be defined conditional on the *impact* of journals for which the student's advisor is an editor. Friedman et al. [6] call any such sequence of references a *slot chain*.

Slot chains essentially define the features used in a probabilistic relational model. When the slot chain also contains one-to-many relationships, then aggregation operators will also be needed to define a feature. Features definable by slot chains can be represented as linear TETs:

$$T_1(v) : \top \overset{w}{-} advisor(v, w) \overset{\emptyset}{-} fame(w) \tag{3}$$

$$T_2(v) : \top \overset{w}{-} advisor(v, w) \overset{u}{-} advisor(u, w), u \neq w \overset{\emptyset}{-} success(u) \tag{4}$$

$T_1(v)$ counts how many famous and how many non-famous advisors entity v has. $T_2(v)$ counts how many successful and non-successful entities u exist that have the same advisor as v.

3.3 Relational Bayesian Networks

Relational Bayesian networks (RBNs) [7, 9] are related both to PRMs and to probabilistic modeling languages based on logic programming (e.g. [18, 11]). RBNs also specify probability distributions over probabilistic relations on a given domain. The representation language is quite different, however: an RBN representation is given by the specification of one *probability formula* for each probabilistic relation. A simple RBN encoding for the domain described in the previous section could be given by assigning to the probabilistic relation *success*(v) the probability formula

$$F(v) = \textit{noisy-or}\{(\textit{fame}(w) : 0.8, 0.5) \mid w : \textit{advisor}(v, w)\}.$$

This formula can be read as a procedural computation rule for computing the probability of a ground atom *success*(a) as follows: determine all objects w for which *advisor*(a, w) is true (which may or may not be exactly one); each such w contributes a value of 0.8 if *fame*(w) is true, and a value of 0.5 if *fame*(w) is false to a multiset of probabilities; finally, compute a single probability value by combining all the probabilities in the multiset with a *noisy-or*.

The reader is referred to [7, 9] for details on syntax and semantics of probability formulas. The preceding small example should be sufficient, however, to illustrate the connection between RBNs and TETs: for the evaluation of the given probability formula for *success*(a) it is sufficient to know the value of $T_1(a)$ with T_1 as in (3). More refined probability formulas could be constructed that define the probability of *success*(a) also dependent on the feature $T_2(a)$. Generally, one can show that for each probability formula $F(\boldsymbol{v})$ one can (automatically) construct a TET $T(\boldsymbol{v})$, such that the evaluation of the probability formula for some objects $\boldsymbol{a}$ only depends on the value $V(T(\boldsymbol{a}))$. On the other hand, the probability formula computes a probability value for each possible value of the TET. Thus, a probability formula is both an (implicit) definition of a TET, and a specification of a discriminant function on that TET.

3.4 Relational Decision Trees

Several approaches exist for adapting decision trees to relational settings [2, 12, 15]. Internal nodes in such a decision tree are labeled by relational features, which can be expressed e.g. in a logical [2, 15] or graphical [12] notation.

Figure 4 shows a relational decision tree described in [15]. This is a decision tree for a domain of web-pages with attributes like *isStudent*,*isFaculty*,*isCourse* describing the nature of a web page (student, faculty, course homepage etc.). Another attribute, *path*, describes whether or not the URL of a page contains a directory path (i.e. pages for which *path*=*false* are top-level pages in a domain).

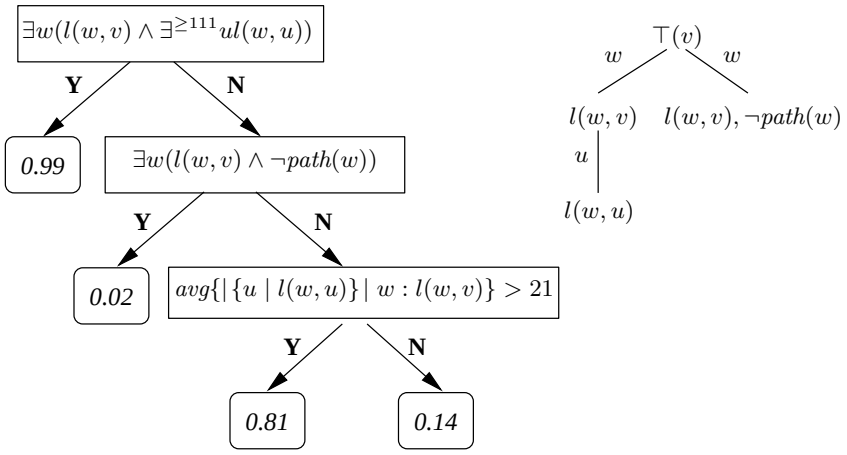


Fig. 4. Relational Decision Tree and Feature TET

The decision tree of figure 4 estimates the probability that a web-page v belongs to the *isStudent* class. For this it tests (at most) three different relational features: at the root node it is tested whether the page v is linked from another page w which has at least 111 outgoing links (indicating, perhaps, a student directory). The second node tests the feature whether v is linked from a page without a directory path in the URL. The third node tests whether the average number of outgoing links from pages w that also contain a link to v is at least 21.

The TET shown on the right of figure 4 represents the combinatorial properties that determine the truth value of the three features tested in the decision tree: for each web page v, the truth value of a feature is a function of $V(T(v))$ (where, in fact, for the truth values of the first and third feature only the value of the left branch is relevant, and for the truth value of the second feature only the right branch is relevant). Furthermore, the probability estimates computed by the decision tree can be defined by a discriminant function on T.

3.5 From Features to Statistics

In standard "propositional" learning settings there is a clear distinction between a *data item* and a *data set*. The latter is a collection of the former, and, moreover, the data items are typically assumed to be independent samples from some common underlying distribution (iid). The distinction between data item and data set gives rise to the distinction between a feature and a *statistic*: the former describes a property of an individual data item, the latter a property of a data set as a whole.

It is well-known that this "iid" model of data is often inappropriate in the context of relational data (see e.g. [10]). Specifically, one can not regard the entities in a relational structure as independent data items.

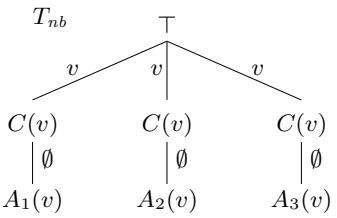


Fig. 5. Naive Bayes Sufficient Statistics

Seeing that in relational data there is no clear dividing line between data items and data sets, there can also not be a clear distinction between a feature and a statistic. Type extension trees provide a clear perspective on both the common nature and the remaining distinction between feature and statistic: a statistic is basically a feature expressed by a TET without free variables. Such a TET represents a global property of a relational structure, without reference to any particular entities in the structure

The following example illustrates the use of TETs to represent global statistics, and how standard propositional data can be treated as a special case of relational structures.

Example 3. Assume a relational signature R containing four unary relation symbols C, A_1, A_2, A_3. A relational structure for R just consists of a collection of entities, and the relations can be read as attributes in the conventional way, i.e. an R-structure can also be seen as a standard propositional dataset.

Consider the TET T_{nb} of figure 5. This is a TET without free variables. Its values represent the 2-way marginal counts of attribute pairs C, A_i $(i = 1, 2, 3)$. Thus, the value of T_{nb} for a given R-structure is just the minimal sufficient statistic for learning a naive Bayes classifier for C given the A_i. Compare this to

$$T_{sat} = \top \overset{v}{-} C(v) \overset{\emptyset}{-} A_1(v) \overset{\emptyset}{-} A_2(v) \overset{\emptyset}{-} A_3(v)$$

Values of T_{sat} are the full joint counts of $C, A_1, \ldots, A_3$, and, thus, are sufficient for learning a saturated model, i.e. without any constraints on the joint distribution of the $C, A_1, \ldots, A_3$.

4 Discussion and Related Work

In the previous sections we have introduced type extension trees as a language for specifying relational features. We have seen that the language is very expressive, and can represent different types of features that have been used within a wide spectrum of different relational learning tasks.

One should compare TETs to two different classes of alternatives. First, we may consider logics, database query or programming languages, which are not specifically designed to specify relational features, but can be employed for that purpose. The advantage of TETs over existing predicate logics is the former's

ability to represent quantitative, combinatorial properties, whereas the latter only define Boolean features. Database query languages like SQL can also retrieve quite complex combinatorial information about the relational neighborhood of an entity. There are several advantages of the TET language over SQL, however: most importantly, the syntax of SQL is much more complex than the syntax of TETs, and does not possess a similarly clearly defined semantics. For example, it is not clear how to distinguish SQL queries that refer to properties of single entities, from queries that refer to entity pairs, entity triples, etc. In TETs this distinction is clearly made by the number of free variables.

Second, TETs need to be compared to specific feature specification languages used in existing learning frameworks. Such specification languages have been defined in terms of e.g. fragments of first-order logic [3], graph fragments [14], and aggregation operators [16]. Most similar in spirit to TETs are perhaps the selection graphs of [12]. However, the latter are much more limited in only being able to express boolean features of single entities. A key advantage of TETs over selection graphs and many other feature languages is their clear parsimonious syntax, coupled with a precise formal semantics.

Apart from being an alternative to existing feature languages, TETs also provide a solid basis for theoretical analyses of their characteristic properties. For example, identifying different feature languages with specific sub-classes of TETs provides a basis for analyzing more clearly what kind of information the different languages are able to extract.

5 Conclusion

We have introduced TETs as an expressive and principled framework for describing features in relational domains. TETs provide a uniform language for specifying features of single entities, tuples of entities, or a relational dataset as a whole. By combining TETs with the notion of discriminant functions one obtains a coherent, integrated view of feature construction and model induction that can capture diverse data mining tasks such as graph mining and statistical relational learning.

So far, we have introduced TETs mostly as a conceptual tool. One prerequisite for using TETs in practice is to complement the language of TETs with a formal language for defining discriminant functions. Based on the TET tree structure it is quite easy to inductively define such languages in a way that is similarly parsimonious and semantically clear as the TET language itself. One example of such a language already exists in the language of probability formulas (section 3.3).

Here we have introduced TETs only for boolean data. However, all our basic definitions can be directly generalized to non-binary data by only relaxing the definition of literals to also include equational literals $r(v) = A$ (A a possible value of categorical attribute r), or $s(v) \geq q$ ($q \in \mathbb{R}$ a threshold value for numerical attribute s).

References

1. A. Blass and Y. Gurevich. Choiceless polynomial time computation and the zero-one law. In *Proc. of CSL 2000*, pages 18–40, 2000.
2. H. Blockeel and L. de Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, (101):285–297, 1998.
3. C. Cumby and D. Roth. Feature extraction languages for propositionalized relational learning. In *Proc. of the IJCAI-2003 workshop on learning statistical models from relational data*, 2003. http://kdl.cs.umass.edu/srl2003.
4. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer Verlag, 1995.
5. Ronald Fagin. Probabilities on finite models. *Journal of Symbolic Logic*, 41(1):50–58, 1976.
6. N. Friedman, Lise Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.
7. M. Jaeger. Relational bayesian networks. In Dan Geiger and Prakash Pundalik Shenoy, editors, *Proceedings of the 13th Conference of Uncertainty in Artificial Intelligence (UAI-13)*, pages 266–273, Providence, USA, 1997. Morgan Kaufmann.
8. M. Jaeger. Convergence results for relational Bayesian networks. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS-98)*, pages 44–55, IEEE Computer Society Press.
9. M. Jaeger. Complex probabilistic modeling with recursive relational Bayesian networks. *Annals of Mathematics and Artificial Intelligence*, 32:179–220, 2001.
10. D. Jensen and J. Neville. Linkage and autocorrelation cause feature selection bias in relational learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML–2002)*, pages 259–266, 2002.
11. K. Kersting and L. De Raedt. Towards combining inductive logic programming and bayesian networks. In *Proceedings of the Eleventh International Conference on Inductive Logic Programming (ILP-2001)*, Springer Lecture Notes in AI 2157, 2001.
12. A. J. Knobbe, A. Siebes, and D. van der Wallen. Multi-relational decision tree induction. In *Proceedings of PKDD-99*, pages 378–383, 1999.
13. Ph. G. Kolaitis and M.Y.Vardi. 0-1 laws and decision problems for fragments of second-order logic. *Information and Computation*, 87:302–338, 1990.
14. S. Kramer and L. de Raedt. Feature construction with version spaces for biochemical applications. In *Proc. of ICML-01*, 2001.
15. J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning relational probability trees. In *Proceedings of SIGKDDD'03*, 2003.
16. C. Perlich and F. Provost. Aggregation-based featrue invention and relational concept classes. In *Proc. of SIGKDD'03*, 2003.
17. A. Pfeffer. *Probabilistic Reasoning for Complex Systems*. PhD thesis, Stanford University, 2000.
18. T. Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP'95)*, pages 715–729, 1995.
19. Takashi Washio and Hiroshi Motoda. State of the art of graph-based data mining. *SIGKDD Explor. Newsl.*, 5(1):59–68, 2003.

Flexible Tree Kernels based on Counting the Number of Tree Mappings

Tetsuji Kuboyama[1], Kilho Shin[2], and Hisashi Kashima[3]

[1] Center for Collaborative Research, University of Tokyo, Japan
4-6-1 Komaba, Meguro, Tokyo, 153-8505, Japan
kuboyama@ccr.u-tokyo.ac.jp
[2] Research Center for Advanced Science and Technology, University of Tokyo, Japan
4-6-1 Komaba, Meguro, Tokyo, 153-8904, Japan
kshin@mpeg.rcast.u-tokyo.ac.jp
[3] Tokyo Research Laboratory, IBM Japan, Ltd., 1623–14 Shimotsuruma, Yamato-shi, Kanagawa, 242–8502,Japan
hkashima@jp.ibm.com

Abstract. Functions counting the number of common sub-patterns between trees have been promising candidates for kernel functions for trees in learning systems. There are several viewpoints of how two patterns between two trees can be regarded as the same. In the tree edit distance, these viewpoints have been well formalized as the class of *tree mappings*, and several distance measures have been proposed according to the classes of tree mappings. In this paper, we address the design problem of new flexible tree kernels corresponding to four representative classes of tree mappings. Therefore, first, we develop four counting functions for tree mappings according to the four classes. Secondly, we prove that three of the four counting functions are kernel functions, and the other is not.

1 Introduction

The kernel method, a method of machine learning, provides a generic framework to address a variety of applications, and is being extensively studied [1].

This paper focuses on kernel methods for trees. In prior work, Collins and Duffy [2] presented the *parse tree kernel* as a counting function of common (subtree) patterns between trees. Kashima and Koyanagi [3] extended the parse tree kernel by allowing elastic structure matching for a more flexible interpretation of the common patterns, which is referred to as an *elastic tree kernel*. However, it has yet to be shown that the counting function really satisfies the required properties of a kernel function, *i.e.* semi-positive definiteness.

For tree edit distance, the problem of what is the common pattern between trees has been thoroughly studied. First, Tai [4] showed that the computation of the tree edit distance can be defined as an optimization problem of a common pattern representation, the *tree mapping*. The tree mapping given by Tai is a

partial node-to-node mapping between two trees that preserves the two fundamental orders defined over the sets of the nodes in the trees (*i.e.* the ancestor-descendent and sibling orders). This tree mapping is referred to as a *Tai mapping*. Inspired by Tai's result, several subclasses of the Tai mapping class have been proposed [5–9], each of which determines a variant of the tree edit distance. Recently, Kuboyama and Shin [10, 11] showed that many of the interesting mapping classes fall into four fundamental classes, which are called the *accordant*, *semi-accordant*, *alignable*, and Tai mappings. The following inclusive relationship holds among these classes.

$$\textsc{Accordant} \subsetneqq \textsc{Semi-Accordant} \subsetneqq \textsc{Alignable} \subsetneqq \textsc{Tai}$$

This class hierarchy represents the flexibility of the interpretation of the common patterns between trees. Note that Tai mapping is the most flexible in this hierarchy.

It is presumable that four tree kernels can be proposed by giving counting functions for tree mappings for these four classes. In fact, the counting function of the elastic tree kernel [3] turned out to be that of the accordant mappings. This fact indicates the possibility that three more flexible tree kernels can be designed, since the elastic tree kernel [3] corresponds to the most restrictive one among the four classes, even though the tree kernel was designed with the intention of designing the most flexible tree kernel possible.

This paper presents new tree kernels more flexible than existing ones by considering the following characteristics.

1. The counting functions for all four of the mapping classes have recursive expressions, which enable efficient evaluation by dynamic programing.
2. The counting functions for the accordant, semi-accordant, and Tai mappings are positive semi-definite. In contrast, for the alignable mapping, a counterexample to positive semi-definiteness exists.
3. Hence, the counting functions for the Tai, semi-accordant, and accordant mappings are kernel functions whereas that for the alignable mapping is not. The accordant case proves that the elastic tree kernel [3] is a kernel function.

This paper is organized as follows. In Sect. 2, tree kernels proposed in prior work and the notion of tree mapping are described. In Sect. 3, the recursive expressions of the counting functions are presented for each class of tree mappings. In Sect. 4, it is shown that the counting functions for the Tai, semi-accordant, and accordant mapping classes are positive semi-definite. For the alignable mapping, a counterexample to positive semi-definiteness is presented. Finally, this paper is concluded in Sect. 5. Most of proofs are omitted due to the space limitation.

2 Background

2.1 Preliminaries

The trees considered in this paper are labeled rooted ordered trees. A labeled tree is a tree in which each node is labeled from a finite alphabet Σ. Let l be a

labeling function which assigns a label from a set $\Sigma = \{a, b, c, \ldots\}$ to each node. An *ordered tree* is a tree in which the left-to-right order among siblings is given. Let $V(T_i)$ denote the set of nodes of a tree T_i $(i = 1, 2)$.

An *ancestor* of a node is recursively defined as follows: the ancestor of a node is either the node itself, or an ancestor of the parent of the node. The notation $x \leq y$ denotes a node y is an ancestor of a node x, and $x \frown y$ denotes the *least* (or *nearest*) *common ancestor* of x and y. If $x \leq y$ and $x \neq y$, then y is a *proper ancestor* of x, and denoted by $x < y$. A node x is *to the left* of a node y if $x \frown y$ is a proper ancestor of both x and y, and the child of $x \frown y$ on the path to x is to the left of the child of $x \frown y$ on the path to y, and denoted by $x \prec y$. In this paper, a 'tree' refers to a labeled rooted ordered tree.

2.2 Tree Kernels

Collins and Duffy [2] presented the *parse tree kernel* as a counting function of common subtrees, which is in the class of convolution kernels [12].

Since a common subtree between T_1 and T_2 uniquely defines a partial mapping between $V(T_1)$ and $V(T_2)$, the parse tree kernel is regarded as a counting function of partial node-to-node mappings between trees. Then the kernel function is basically defined as follows. (Note that the definition is slightly modified from the original.)

$$\mathbf{K}(T_1, T_2) = \sum_{v_1 \in V(T_1)} \sum_{v_2 \in V(T_2)} \mathbf{K}^{\mathrm{SC}}(v_1, v_2),$$

where $\mathbf{K}^{\mathrm{SC}}(v_1, v_2)$ is the counting function of the number of the partial mappings f that satisfy the following conditions.

- f is a mapping from a domain $\mathcal{D} \subseteqq V(T_1)$ to $V(T_2)$ with $f(v_1) = v_2$.
- Any $v \in \mathcal{D} \setminus \{v_1\}$ and its parent w in T_1 satisfy: (1) v is a descendent of v_1; (2) $w \in \mathcal{D}$; (3) w and $f(w)$ have the same number of children; and (4) If v is the i-th child of w, then $f(v)$ is also the i-th child of $f(w)$.

A partial mapping satisfying the above conditions is referred to as a *subtree-congruent* mapping. The value of $\mathbf{K}^{\mathrm{SC}}(v_1, v_2)$ can be calculated by the following equality.

$$\mathbf{K}^{\mathrm{SC}}(v_1, v_2) = \begin{cases} \delta_{l(v_1), l(v_2)} \cdot \prod_{i=1}^{\#\mathrm{ch}(v_1)} (1 + \mathbf{K}^{\mathrm{SC}}(v_{1i}, v_{2i})) & \text{if } \#\mathrm{ch}(v_1) = \#\mathrm{ch}(v_2); \\ 0 & \text{otherwise} \end{cases}$$

where by $\#\mathrm{ch}(v)$, v_{ij}, and $l(v_i)$, we denote the number of the children of v, the j-th child of v_i, and the label attached to v_i respectively; and δ_{ij} is Kronecker's delta, *i.e.* $\delta_{ij} = 1$ if $i = j$ and 0 otherwise.

The value of the kernel $\mathbf{K}(T_1, T_2)$ can be calculated by dynamic programming in $O(|V(T_1)| \cdot |V(T_2)|)$ time. Note that the subtree-congruent mapping is injective and preserves the ancestor-descendant and the sibling orders between two trees.

Kashima and Koyanagi [3] thought that the subtree-congruent mapping is too restrictive for HTML parse trees, and proposed an extended tree kernel, the *elastic tree kernel*, based on a more flexible mapping. Unfortunately, their work is incomplete for two reasons.

– The function has yet to be shown to be positive semi-definite.
– The elastic tree kernel turns out to be a counting function of a relatively restrictive class of tree mappings in the study of the tree edit distance. This fact implies that the kernel is not as flexible as they intended.

This paper resolves these problems.

2.3 Common Patterns and Tree Mappings in the Tree Edit Distance

Tree Edit Distance and Tree Mappings. The edit distance from a tree S to a tree T is defined as the minimum cost of an *edit sequence* that transforms S to T. An edit sequence is a sequence of the primitive edit operations of (1) relabeling, (2) deleting, and (3) inserting a node in a tree. The edit operation of changing the label of a node from a to b $(a, b \in \Sigma)$ is denoted by $\langle a \to b \rangle$, and the edit operations of deleting and inserting a node with label a are respectively denoted by $\langle a \to \lambda \rangle$ and $\langle \lambda \to a \rangle$, where λ denotes the null symbol. The cost of the operation $\langle \alpha \to \beta \rangle$, where $\alpha, \beta \in \Sigma \cup \{\lambda\}$, is denoted by $\gamma(\alpha \to \beta)$ and is assumed to satisfy $\gamma(\alpha \to \beta) \leq \gamma(\alpha \to \mu) + \gamma(\mu \to \beta)$ for arbitrary $\alpha, \beta, \mu \in \Sigma \cup \{\lambda\}$.

Tai [4] showed a fundamental correspondence between an effect of an edit sequence and a common pattern occurring in two trees. The common pattern is represented as a set of pairs of nodes called a *tree mapping*. The formal definition of the tree mapping is given as follows.

Definition 1. *$M \subseteqq V(S) \times V(T)$ is said to be a tree mapping, if and only if the following are satisfied for arbitrary $(s_1, t_1), (s_2, t_2) \in M$. (1) $s_1 = s_2 \Leftrightarrow t_1 = t_2$, (2) $s_1 < s_2 \Leftrightarrow t_1 < t_2$, and (3) $s_1 \prec s_2 \Leftrightarrow t_1 \prec t_2$.*

The tree mapping defined by Tai is referred to as a *Tai mapping*. An example of a Tai mapping is shown in Fig. 1. With the notation of $M_S = \{s \mid (s, t) \in M\}$ and $M_T = \{t \mid (s, t) \in M\}$, the cost of the Tai mapping M is defined by

$$\gamma(M) = \sum_{(s,t) \in M} \gamma(l(s) \to l(t)) + \sum_{s \notin M_S} \gamma(l(s) \to \lambda) + \sum_{t \notin M_T} \gamma(\lambda \to l(t)).$$

Tai [4] showed that the tree edit distance is defined as the cost minimization problem of tree mappings as follows.

$$d(S, T) = \min\{\gamma(M) \mid M \text{ is a Tai mapping from } S \text{ to } T\} \tag{1}$$

In the rest of this paper, the problem of calculating the right-hand side of Eq. (1) is referred to as the edit problem.

Class Hierarchy of Tree Mappings. In the tree edit distance, several classes of tree mappings have been proposed by restricting the conditions for the Tai mapping. There are two major motivations for restricting the Tai mapping. The first is to improve the computation cost of the edit problem. The second is to tailor the tree mapping for specific applications, since the Tai mapping may be too general for certain applications such as comparing parse trees, taxonomies, and so forth. The following are some of the most important classes of tree mappings in the tree edit distance.

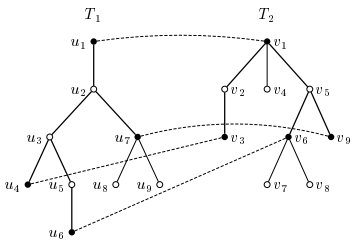


Fig. 1. Example of a Tai mapping: $M = \{(u_1, v_1), (u_4, v_3), (u_6, v_6), (u_7, v_9)\}$

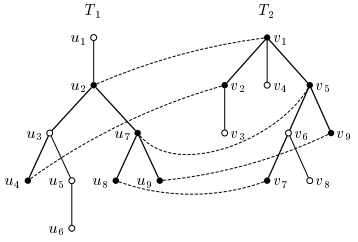


Fig. 2. Example of an accordant mapping: $M = \{(u_2, v_1), (u_4, v_2), (u_7, v_5), (u_8, v_7), (u_9, v_9)\}$

Table 1. Characteristic of Mapping Classes

T_1 / T_2	s_1 s_2 s_3	s_1 s_2 s_3	s_1 s_2 s_3	s_1 s_2 s_3	s_1 s_2 s_3
t_1 t_2 t_3	Tai, Al S-Ac, Ac	Tai, Al	Tai	none	none
t_1 t_2 t_3	Tai, Al	Tai, Al S-Ac, Ac	Tai, Al	none	none
t_1 t_2 t_3	Tai	Tai, Al	Tai, Al S-Ac, Ac	none	none
t_1 t_2 t_3	none	none	none	Tai, Al S-Ac, Ac	Tai, Al S-Ac
t_1 t_2 t_3	none	none	none	Tai, Al S-Ac	Tai, Al S-Ac, Ac

Al: Alignable, S-Ac: Semi-Accordant, Ac: Accordant; $M = \{(s_1, t_1), (s_2, t_2), (s_3, t_3)\}$.

- Zhang [5] introduced the notion of the *constrained* mapping and proposed a quadratic time algorithm for the edit problem for the constrained mapping.
- Lu et al. [6] slightly relaxed the definition of the constrained tree mapping and gave the notion of the *less-constrained* mapping.
- Richter [7] introduced the *structure-respecting* mapping for comparing syntax trees.
- Jiang et al. [8] extended the alignment of sequences to the alignment of trees, and Kuboyama and Shin [10] elucidated the mapping definition for the alignment of trees called an *alignable* mapping.

Recently, Kuboyama and Shin [10, 11] showed the following inclusive relation.

$$\textsc{Constrained} = \textsc{Structure-Respecting} \subsetneqq \textsc{Less-Constrained} = \textsc{Alignable}$$

In this paper, the constrained and structure-respecting mappings are both referred to as *semi-accordant* mapping, and the less-constrained and alignable mappings are both referred to as *alignable* mappings. In addition, Shin and Kuboyama [11] introduced the notion of the *accordant* mapping. The following are the formal definitions of these mapping classes.

Accordant mapping. A Tai mapping M is *accordant* iff $s_1 \smile s_2 = s_1 \smile s_3 \Leftrightarrow t_1 \smile t_2 = t_1 \smile t_3$ holds for any $(s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$.

Semi-Accordant mapping. A Tai mapping M is *semi-accordant* iff $s_1 \smile s_2 = s_1 \smile s_3 \Leftrightarrow t_1 \smile t_2 = t_1 \smile t_3$ holds for any $(s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$ such that none of s_1, s_2, or s_3 is an ancestor of any of the others.

Alignable mapping. A Tai mapping M is *alignable* iff $s_1 \smile s_2 < s_1 \smile s_3 \Rightarrow t_1 \smile t_3 = t_2 \smile t_3$ holds for any $(s_1, t_1), (s_2, t_2), (s_3, t_3) \in M$.

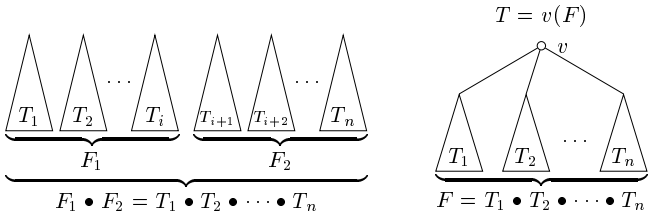


Fig. 3. The composition of two forests F_1 and F_2, and a tree $T = v(F)$.

An example of an accordant mapping is shown in Fig. 2. Each class is characterized by the properties depicted in Tab. 1. It is easy to show that the elastic tree kernel [3] turns out to be the counting function of the accordant mappings between two trees.

3 Recursive Expressions of Counting Functions

In this section, first, we extend the notion of tree mappings to *forests*. Next, we give the counting functions of the extended tree mappings between two forests.

3.1 Mapping Definition for Forests

A forest F is an ordered sequence of disjoint trees $T_1, \ldots, T_n$, denoted by $T_1 \bullet \cdots \bullet T_n$, or $\bullet_{i=1}^{n} T_i$, and in particular denoted by $\emptyset$ if $n = 0$. The composition of forests F_1 and F_2 in this order is denoted by $F_1 \bullet F_2$ (See Fig. 3). The set of nodes $V(T_1) \cup \cdots \cup V(T_n)$ is denoted by $V(F)$.

Definition 2. *Let F be a forest $T_1 \bullet \cdots \bullet T_n$ and v be a node not included in $V(F)$. The tree $v(F)$ is defined as follows.*

- $V(v(F)) = (\bigcup_{i=1}^{n} V(T_i)) \cup \{v\}$.
- $x < y$ *iff, for some i, either $x, y \in V(T_i)$ and $x < y$ in T_i, or $y = v$ holds.*
- $x \prec y$ *holds iff, for some i, either $x, y \in V(T_i)$ and $x \prec y$ in T_i or $x \in V(T_i)$, $y \in V(T_j)$ and $i < j$ holds.*

The notion of tree mappings is extended to forests as follows.

Definition 3. *Let F_1 and F_2 be forests and $\mathcal{C}$ denote one of* Accordant, Semi-Accordant, Alignable, *or* Tai. *A non-empty mapping $M \subseteq V(F_1) \times V(F_2)$ is said to be a $\mathcal{C}$-mapping if and only if M defines a $\mathcal{C}$-mapping from $v_1(F_1)$ to $v_2(F_2)$.*

Let $\mathcal{M}^{\mathcal{C}}(F_1, F_2)$ denote the set of $\mathcal{C}$-mappings between F_1 and F_2. The counting functions $\mathbf{K}^{\mathcal{C}}(F_1, F_2)$ are defined as follows, where the symmetric function $\sigma : \Sigma \times \Sigma \to \mathbb{R}_0^+$ defines the similarity between labels of nodes, where $\mathbb{R}_0^+$ denotes the set of all non-negative real numbers.

$$\mathbf{K}^{\mathcal{C}}(F_1, F_2) = \sum_{M \in \mathcal{M}^{\mathcal{C}}(F_1, F_2)} \sigma(M); \quad \sigma(M) = \prod_{(x_1, x_2) \in M} \sigma(l(x_1), l(x_2))$$

In the rest of this paper, a forest F' is called a subforest of a forest F if and only if $V(F')$ is a subset of $V(F)$ and the ancestor-descendant and sibling orders of F' are inherited from F.

3.2 Counting Function for Tai Mapping Class

$$
\begin{aligned}
&\mathbf{K}^{\text{TAI}}(F, \emptyset) = \mathbf{K}^{\text{TAI}}(\emptyset, F) = 0 \\
&\mathbf{K}^{\text{TAI}}(v_1(F_1') \bullet F_1'', v_2(F_2') \bullet F_2'') = \\
&\quad \sigma(l(v_1), l(v_2))(1 + \mathbf{K}^{\text{TAI}}(F_1', F_2'))(1 + \mathbf{K}^{\text{TAI}}(F_1'', F_2'')) \\
&\quad + \mathbf{K}^{\text{TAI}}(v_1(F_1') \bullet F_1'', F_2' \bullet F_2'') + \mathbf{K}^{\text{TAI}}(F_1' \bullet F_1'', v_2(F_2') \bullet F_2'') \\
&\quad - \mathbf{K}^{\text{TAI}}(F_1' \bullet F_1'', F_2' \bullet F_2'') \qquad (2)
\end{aligned}
$$

The natural properties of the Tai mapping (Prop. 1 and Lem. 1) play a central role in proving the correctness of Eq. (2). Let $M' = M \cap (V(F_1') \times V(F_2'))$ and $M'' = M \cap (V(F_1'') \times V(F_2''))$ in the following proposition and lemma.

Proposition 1. *Let F_i' be any subforest of F_i. Then the following hold.*

1. *If M is a Tai mapping from F_1 to F_2, then M' is also a Tai mapping from F_1' to F_2'.*
2. *For non-empty $M \subseteq V(F_1') \times V(F_2')$, the following two properties are equivalent.*
   (a) M is a Tai mapping from F_1' to F_2'.
   (b) M is a Tai mapping from F_1 to F_2.

Lemma 1. *Let F_i' and F_i'' be completely distinct. For non-empty $M \subseteq (V(F_1') \cup V(F_1'')) \times (V(F_2') \cup V(F_2''))$, the following two conditions are equivalent.*

1. *$M \cup \{(v_1, v_2)\}$ is a Tai mapping from $v_1(F_1') \bullet F_i''$ to $v_2(F_2') \bullet F_2''$.*
2. *M satisfies the following three conditions. (1) $M = M' \cup M''$; (2) M' is a Tai mapping from F_1' to F_2'; and (3) M'' is a Tai mapping from F_1'' to F_2''.*

The left-hand side of Eq.(2) is decomposed into the following two parts.

$$\mathbf{K}^{\text{TAI}}(v_1(F_1') \bullet F_1'', v_2(F_2') \bullet F_2'') = \textstyle\sum_{(v_1,v_2) \in M} \sigma(M) + \sum_{(v_1,v_2) \notin M} \sigma(M)$$

Assume $(v_1, v_2) \in M$. The mapping $M \setminus \{(v_1, v_2)\}$ is identical to $M' \cup M''$ for some Tai mappings M' from F_1' to F_2' and M'' from F_1'' to F_2'' by Lem. 1. To the contrary, for arbitrary Tai mappings M' from F_1' to F_2' and M'' from F_1'' to F_2'', $M' \cup M'' \cup \{(v_1, v_2)\}$ is also a Tai mapping from $v_1(F_1') \bullet F_i''$ to $v_2(F_2') \bullet F_2''$ by Lem. 1.

Therefore, since $\sigma(M) = \sigma(l(v_1), l(v_2))\sigma(M')\sigma(M'')$, the following holds.

$$\textstyle\sum_{(v_1,v_2) \in M} \sigma(M) = \sigma(l(v_1), l(v_2))(1 + \mathbf{K}^{\text{TAI}}(F_1', F_2'))(1 + \mathbf{K}^{\text{TAI}}(F_1'', F_2''))$$

Let S_1, S_2, and S_3 be as follows.

$$S_1 = \{M \in \mathcal{M}^{\text{TAI}}(v_1(F_1') \bullet F_1'', v_2(F_2') \bullet F_2'') \mid (v_1, w) \in M \wedge w \neq v_2\},$$

$$S_2 = \{M \in \mathcal{M}^{\text{TAI}}(v_1(F_1') \bullet F_1'', v_2(F_2') \bullet F_2'') \mid (w, v_2) \in M \wedge w \neq v_1\}, \text{ and}$$

$$S_3 = \{M \in \mathcal{M}^{\text{TAI}}(v_1(F_1') \bullet F_1'', v_2(F_2') \bullet F_2'') \mid (v, w) \in M \Rightarrow v \neq v_1 \wedge v \neq v_2\}.$$

By condition 2 of Prop. 1, the following holds.

$$\textstyle\sum_{M \in S_1} \sigma(M) = \mathbf{K}^{\text{TAI}}(v_1(F_1') \bullet F_1'', F_2' \bullet F_2'') - \mathbf{K}^{\text{TAI}}(F_1' \bullet F_1'', F_2' \bullet F_2''),$$
$$\textstyle\sum_{M \in S_2} \sigma(M) = \mathbf{K}^{\text{TAI}}(F_1' \bullet F_1'', v_2(F_2') \bullet F_2'') - \mathbf{K}^{\text{TAI}}(F_1' \bullet F_1'', F_2' \bullet F_2''), \text{ and}$$
$$\textstyle\sum_{M \in S_3} \sigma(M) = \mathbf{K}^{\text{TAI}}(F_1' \bullet F_1'', F_2' \bullet F_2'').$$

The correctness of Eq. (2) is shown by adding together all of the above components.

3.3 Template of Counting Function for Subclasses of Tai Mappings

Since Prop. 1 and Lem. 1 do not hold for the subclasses of the Tai mapping, then Eq. (2) is not applicable. However, there exists a common template of counting functions applicable to all three of the subclasses.

$$\mathcal{C} \in \{\textsc{Accordant}, \textsc{Semi-Accordant}, \textsc{Alignable}\}$$

$$\mathbf{K}^{\mathcal{C}}(\emptyset, F) = \mathbf{K}^{\mathcal{C}}(F, \emptyset) = \mathbf{K}^{\mathcal{C}}_{\mathrm{f}}(T, \emptyset) = 0 \tag{4}$$

$$\mathbf{K}^{\mathcal{C}}(T_1 \bullet F_1, T_2 \bullet F_2) = \mathbf{K}^{\mathcal{C}}_{1}(T_1 \bullet F_1, T_2 \bullet F_2) + \mathbf{K}^{\mathcal{C}}(F_1, F_2) \tag{5}$$

$$\begin{aligned}\mathbf{K}^{\mathcal{C}}_{\mathrm{f}}(v(F_1), T_2 \bullet F_2) = {} & \mathbf{K}^{\mathcal{C}}_{\mathrm{t}}(v(F_1), T_2) - \mathbf{K}^{\mathcal{C}}_{\mathrm{f}}(T_2, F_1) + \mathbf{K}^{\mathcal{C}}_{\mathrm{f}}(v(F_1), F_2) \\ & - \mathbf{K}^{\mathcal{C}}(F_1, F_2) + \mathbf{K}^{\mathcal{C}}(F_1, T_2 \bullet F_2)\end{aligned} \tag{6}$$

$$\begin{aligned}\mathbf{K}^{\mathcal{C}}_{\mathrm{t}}(v_1(F_1), v_2(F_2)) = {} & \sigma(l(v_1), l(v_2)) \cdot (1 + \mathbf{K}^{\mathcal{C}}_{2}(F_1, F_2)) + \mathbf{K}^{\mathcal{C}}_{\mathrm{f}}(v_1(F_1), F_2) \\ & + \mathbf{K}^{\mathcal{C}}_{\mathrm{f}}(v_2(F_2), F_1) - \mathbf{K}^{\mathcal{C}}(F_1, F_2)\end{aligned} \tag{7}$$

$\mathbf{K}^{\mathcal{C}}_{1}(F_1, F_2)$ and $\mathbf{K}^{\mathcal{C}}_{2}(T_1, T_2)$ in the template are defined as follows.

$$\mathbf{K}^{\mathcal{C}}_{1}(T_1 \bullet F_1, T_2 \bullet F_2) = \sum_{M \in S_1} \sigma(M); \quad \mathbf{K}^{\mathcal{C}}_{2}(v_1(F_1), v_2(F_2)) = \sum_{M \in S_2} \sigma(M),$$

where

$$S_1 = \{M \in \mathcal{M}^{\mathcal{C}}(T_1 \bullet F_1, T_2 \bullet F_2) \mid M \cap (V(T_1) \times V(T_2 \bullet F_2)) \neq \emptyset \vee M \cap (V(T_1 \bullet F_1) \times V(T_2)) \neq \emptyset\}, \text{ and}$$

$$S_2 = \{M \in \mathcal{M}^{\mathcal{C}}(v_1(F_1), v_2(F_2)) \mid M \cup \{(v_1, v_2)\} \in \mathcal{M}^{\mathcal{C}}(v_1(F_1), v_2(F_2))\}.$$

The recursive expressions for $\mathbf{K}^{\mathcal{C}}_{1}(F_1, F_2)$ and $\mathbf{K}^{\mathcal{C}}_{2}(F_1, F_2)$ are given afterward. $\mathbf{K}^{\mathcal{C}}_{\mathrm{t}}(T_1, T_2)$ takes two trees as the arguments, and $\mathbf{K}^{\mathcal{C}}_{\mathrm{f}}(T, F)$ takes a tree and a forest. Equations (5) to (7) are verified in a similar way to Eq. (2), and the following Prop. 2 is used instead of Prop. 1 and Lem. 1.

Proposition 2. *Let $\mathcal{C}$ be one of* Accordant, Semi-Accordant, *or* Alignable. *Further, let F'_i be a subforest of F_i satisfying one of the following. (1) $F_i = F'_i$; (2) $F_i = v_i(F'_i)$; or (3) $F_i = G_i \bullet F'_i \bullet H_i$ for some forests G_i, H_i. Then the following hold.*

1. *For a $\mathcal{C}$-mapping M from F_1 to F_2, $M' = M \cap (V(F'_1) \times V(F'_2))$ is a $\mathcal{C}$-mapping from F'_1 to F'_2.*
2. *For arbitrary $M \subseteq V(F'_1) \times V(F'_2)$, the following two properties are equivalent. (1) M is a $\mathcal{C}$-mapping from F'_1 to F'_2; and (2)M is a $\mathcal{C}$-mapping from F_1 to F_2.*

In the rest of this section, the following notations are used for $F_i = \bullet_{j=1}^{n_i} T_j^i \quad (i = 1, 2)$ and $M \in \mathcal{M}^{\mathcal{C}}(F_1, F_2)$: $M_j^1 = M \cap (V(T_j^1) \times V(F_2))$, $M_j^2 = M \cap (V(F_1) \times V(T_j^2))$, and $M_{ij} = M_i^1 \cap M_j^2$.

The expressions of counting functions are stated without further proof of their correctness due to the space limitation.

Expressions for $\mathbf{K}_1^{\mathbf{Acc}}$ and $\mathbf{K}_1^{\mathbf{Semi\text{-}Acc}}$

$$\begin{aligned}
&\mathcal{C} \in \{\textsc{Accordant}, \textsc{Semi-Accordant}\}\\
&\mathbf{K}_1^{\mathcal{C}}(T_1 \bullet F_1, T_2 \bullet F_2) = \mathbf{K}_{\mathrm{t}}^{\mathcal{C}}(T_1, T_2)(\mathbf{K}_3^{\mathcal{C}}(F_1, F_2) - 1)\\
&\quad + \mathbf{K}_{\mathrm{f}}^{\mathcal{C}}(T_1, T_2 \bullet F_2) - \mathbf{K}_{\mathrm{f}}^{\mathcal{C}}(T_1, F_2) + \mathbf{K}_{\mathrm{f}}^{\mathcal{C}}(T_2, T_1 \bullet F_1) - \mathbf{K}_{\mathrm{f}}^{\mathcal{C}}(T_2, F_1)\\
&\quad + \mathbf{K}^{\mathcal{C}}(T_1 \bullet F_1, F_2) + \mathbf{K}^{\mathcal{C}}(F_1, T_2 \bullet F_2) - 2\mathbf{K}^{\mathcal{C}}(F_1, F_2) \qquad (8)
\end{aligned}$$

$\mathbf{K}_3^{\mathcal{C}}$ is defined as

$\mathbf{K}_3^{\mathcal{C}}(F_1, F_2) = \sum_{M \in S_3} \sigma(M)$,

where

$$S_3 = \{M \in \mathcal{M}^{\mathcal{C}}(F_1, F_2) \mid M_{ij} \neq \emptyset \wedge M_{kl} \neq \emptyset \Rightarrow (i,j) = (k,l) \vee (i \neq j \wedge k \neq l)\}.$$

$$\begin{aligned}
&\mathcal{C} \in \{\textsc{Accordant}, \textsc{Semi-Accordant}\}\\
&\mathbf{K}_3^{\mathcal{C}}(F, \emptyset) = \mathbf{K}_3^{\mathcal{C}}(\emptyset, F) = 0\\
&\mathbf{K}_3^{\mathcal{C}}(T_1 \bullet F_1, T_2 \bullet F_2) = \mathbf{K}_{\mathrm{t}}^{\mathcal{C}}(T_1, T_2)(1 + \mathbf{K}_3^{\mathcal{C}}(F_1, F_2))\\
&\qquad + \mathbf{K}_3^{\mathcal{C}}(F_1, T_2 \bullet F_2) + \mathbf{K}_3^{\mathcal{C}}(T_1 \bullet F_1, F_2) - \mathbf{K}_3^{\mathcal{C}}(F_1, F_2) \qquad (9)
\end{aligned}$$

Expressions for $\mathbf{K}_1^{\mathbf{Algn}}$ and $\mathbf{K}_2^{\mathbf{Acc}}$

$$\begin{aligned}
&\mathbf{K}_1^{\textsc{Algn}}(\bullet_{i=1}^m T_i^1, \bullet_{j=1}^n T_j^2) = \kappa_{1,1}^{1,1}(1 + \kappa_{2,n}^{2,m})\\
&\quad + \textstyle\sum_{j=2}^n \{(\kappa_{1,j}^{1,1} - \kappa_{1,j-1}^{1,1})(1 + \kappa_{j+1,n}^{2,m})\\
&\qquad + \textstyle\sum_{i=2}^m (\kappa_{j,j}^{1,i} - \kappa_{j,j}^{1,i-1} - \kappa_{j,j}^{2,i} + \kappa_{j,j}^{2,i-1})(1 + \kappa_{j+1,n}^{i+1,m})\}\\
&\quad + \textstyle\sum_{i=2}^m \{(\kappa_{1,1}^{1,i} - \kappa_{1,1}^{1,i-1}(1 + \kappa_{2,n}^{i+1,m})\\
&\qquad + \textstyle\sum_{j=2}^n (\kappa_{1,j}^{i,i} - \kappa_{1,j-1}^{i,i} - \kappa_{2,j}^{i,i} + \kappa_{2,j-1}^{i,i})(1 + \kappa_{j+1,n}^{i+1,m})\} \qquad (10)
\end{aligned}$$

Let a, b, c, and d satisfy $1 \leq a \leq b \leq n_1$ and $1 \leq c \leq d \leq n_2$.

$$\kappa_{c,d}^{a,b} = \mathbf{K}^{\textsc{Algn}}(\bullet_{i=a}^b T_i^1, \bullet_{j=c}^d T_j^2) \qquad (11)$$

$$\mathbf{K}_2^{\textsc{Acc}}(F_1, F_2) = \mathbf{K}_3^{\textsc{Acc}}(F_1, F_2) \qquad (12)$$

Expressions for $\mathbf{K}_2^{\mathbf{Semi\text{-}Acc}}$ and $\mathbf{K}_2^{\mathbf{Algn}}$

$$\begin{aligned}
&\mathcal{C} \in \{\textsc{Semi-Accordant}, \textsc{Alignable}\}\\
&\mathbf{K}_2^{\mathcal{C}}(F_1, F_2) = \mathbf{K}^{\mathcal{C}}(F_1, F_2) \qquad (13)
\end{aligned}$$

The left-to-right recursive evaluation of Eq. (2) to Eq. (13) actually terminate due to the base expression in Eq. (4). Each counting function can be evaluated with the time complexity $O(n^4)$ for the Tai, $O(n^2d^2)$ for the alignable, $O(n^2)$ for the semi-accordant, and $O(n^2)$ for the accordant mapping class, where n denotes the size of trees, and d denotes the maximum degree by dynamic programing as in the case of the algorithms for the tree edit distance.

Remark. The counting functions in the parse tree kernel and the elastic tree kernel both require a pair of nodes (s_r, t_r) in the mapping M such that $s_r \geq s$ and $t_r \geq t$ for any $(s,t) \in M$. It is easy to modify our counting functions proposed in this paper to satisfy this restriction. Nevertheless, we believe our counting functions are more natural as the similarity measures between two trees.

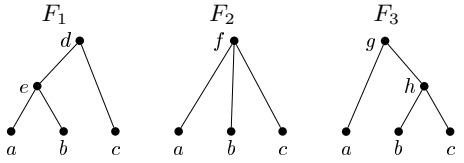
4 Positive Semi-Definiteness of Counting Functions

Assume that the label-similarity function σ is positive semi-definite. Then intuition may suggest that the positive semi-definiteness of $\mathbf{K}^{\mathcal{C}}$ can be inferred from the fact that $\mathbf{K}^{\mathcal{C}}(x,y)$ can be represented as a polynomial in $\sigma(a,b)$.

However, this intuition is incorrect when $\mathcal{C}$ is ALIGNABLE. Consider the three forests $F_1, F_2, and F_3$ and the label-similarity function σ over $\Sigma = \{a,b,c,d,e,f,g,h\}$ as shown in Fig. 4. The Gram matrix $[\mathbf{K}^{\text{ALGN}}(F_i,F_j)]$ is given by:

$$[\mathbf{K}^{\text{ALGN}}(F_i,F_j)] = \begin{bmatrix} 7+16\epsilon+8\epsilon^2 & 7 & 6 \\ 7 & 7+8\epsilon & 7 \\ 6 & 7 & 7+16\epsilon+8\epsilon^2 \end{bmatrix}$$

Since its determinant D coincides with $-7+\epsilon \cdot q(\epsilon)$ for some quartic $q(\epsilon)$, D is negative for a sufficiently small $\epsilon < 1$ ($\epsilon > 0$), and therefore, the matrix has at least one negative eigenvalue. This fact means that $\mathbf{K}^{\text{ALGN}}(x,y)$ is not a kernel function.



$$\sigma(x,y) = \begin{cases} 1 & \text{if } x = y \in \{a,b,c\} \\ \epsilon & \text{if } x = y \in \{d,e,f,g,h\} \\ 0 & \text{if } x \neq y \end{cases}$$

Fig. 4. A counterexample to positive semi-definiteness

In contrast, $\mathbf{K}^{\mathcal{C}}$ is positive semi-definite when $\mathcal{C}$ is one of TAI, SEMI-ACCORDANT, or ACCORDANT (Cor. 1). Prop. 3 plays a key role to prove Cor. 1.

In Prop. 3, the following notations are used.

- When A^{ij} are m-dimensional square matrices parameterized by $(i,j) = \{1,\ldots,n\}^2$, A denotes the derived mn-dimensional square matrix $[A^{ij}]_{i,j=1,\ldots,n}$; and the $(mi+k, mj+l)$-element of A is defined to be the (k,l)-element of A^{ij}, and denoted by A^{ij}_{kl}.
- p is a homogeneous polynomial of degree d in the m^2 variables $X_{11}, X_{12}, \ldots, X_{mm}$. Further, assume that p is given a representation of

$$p(X_{11}, X_{12}, \ldots, X_{mm}) = \sum_{\boldsymbol{k} \in \{1,\ldots,m\}^d} \sum_{\boldsymbol{l} \in \{1,\ldots,m\}^d} c_{\boldsymbol{k},\boldsymbol{l}} X_{k_1 l_1} \cdot \cdots \cdot X_{k_d l_d},$$

  where $\boldsymbol{k} = (k_1,\ldots,k_d)$ and $\boldsymbol{l} = (l_1,\ldots,l_d)$. Note that such representation of p is not unique.

Proposition 3. *Let A be positive semi-definite. If there exists $\bar{c}_{\boldsymbol{k}} \in \mathbb{R}$ for each $\boldsymbol{k} \in \{1, .., m\}^d$ such that $c_{\boldsymbol{k},\boldsymbol{l}} = \bar{c}_{\boldsymbol{k}}\bar{c}_{\boldsymbol{l}}$, then the n-dimensional square matrix $[p(A^{ij}_{11}, \ldots, A^{ij}_{mm})]_{i,j=1,\ldots,n}$ is also positive semi-definite.*

For a positive integer N, let $\mathcal{F}_N$ denote the set of forests F such that $|V(F)| \leq N$. A *universal tree* $\mathcal{T}_{\mathcal{N}}$ is a tree with a finite set of nodes, into which each forest $F \in \mathcal{F}_N$ is embedded preserving the ancestor-descendent and sibling orders. When a single order-preserving embedding $e_F : V(F) \to V(\mathcal{T}_{\mathcal{N}})$ is assigned to each $F \in \mathcal{F}_N$, a pair of $\mathcal{T}_{\mathcal{N}}$ and the set $\{e_F \mid F \in \mathcal{F}_N\}$ is used as a common numbering scheme of the nodes for any $F \in \mathcal{F}_N$.

Let $\mathcal{C}$ denote an arbitrary subclass of the Tai mapping, including, but not limited to, Tai, Alignable, Semi-Accordant, and Accordant. The only restriction imposed on $\mathcal{C}$ is to satisfy $\{(v_1, v_1), \ldots, (v_n, v_n)\} \in \mathcal{M}^{\mathcal{C}}(F, F)$ for arbitrary F, n and $v_1, \ldots, v_n \in V(F)$, where $\mathcal{M}^{\mathcal{C}}(F_1, F_2)$ denotes the set of $\mathcal{C}$-mappings from F_1 to F_2.

The mapping class $\mathcal{C}$ is said to be *absorbent* if and only if, for an arbitrary N, there exists a pair of a universal tree $\mathcal{T}_{\mathcal{N}}$ and a set of embedding $\{e_F \mid F \in \mathcal{F}_N\}$ such that: $\forall(F_1 \in \mathcal{F}_N)\forall(F_2 \in \mathcal{F}_N)\forall(M \in V(F_1) \times V(F_2))[M \in \mathcal{M}^{\mathcal{C}}(F_1, F_2) \Leftrightarrow (e_{F_1} \times e_{F_2})(M) \in \mathcal{M}^{\mathcal{C}}(\mathcal{T}_{\mathcal{N}}, \mathcal{T}_{\mathcal{N}})]$. In addition, $\mathcal{C}$ is said to be *transitive* if and only if, for arbitrary $M_{12} \in \mathcal{M}^{\mathcal{C}}(F_1, F_2)$ and $M_{23} \in \mathcal{M}^{\mathcal{C}}(F_2, F_3)$, the mapping $M_{13} = \{(v_1, v_3) \mid \exists(v_2)[(v_1, v_2) \in M_{12} \wedge (v_2, v_3) \in M_{23}]\}$ is also a $\mathcal{C}$-mapping from F_1 to F_3.

If $\mathcal{C}$ is transitive, the inverse of a given $\mathcal{C}$-mapping and the composition of given $\mathcal{C}$-mappings are all $\mathcal{C}$-mappings. In particular, $\mathcal{M}^{\mathcal{C}}(F, F)$ under map composition forms a group.

Theorem 1. *Let $\sigma : \Sigma \times \Sigma \to \mathbb{R}^+_0$ be positive semi-definite. If $\mathcal{C}$ is absorbent and transitive, the function $\mathbf{K}^{\mathcal{C}}|_{\mathcal{F}_N} : \mathcal{F}_N \times \mathcal{F}_N \to \mathbb{R}^+_0$ is positive semi-definite for an arbitrary N.*

It is easy to see that Accordant, Semi-Accordant, Alignable and Tai classes are all absorbent. In contrast, the alignable mapping class is not transitive, while the others are. A counterexample to the transitivity of the alignable mapping is also given by Fig. 4: $M_{12} = \{(a, a), (b, b), (c, c)\} \in \mathcal{M}^{\text{Algn}}(F_1, F_2)$; $M_{23} = \{(a, a), (b, b), (c, c)\} \in \mathcal{M}^{\text{Algn}}(F_2, F_3)$; $M_{13} = \{(a, a), (b, b), (c, c)\} \notin \mathcal{M}^{\text{Algn}}(F_1, F_3)$. Note that, for simplicity, in the mappings, labels are used to indicate the nodes. Corollary 1 gives the main assertion of this section.

Corollary 1. *Let $\mathcal{C}$ be one of* Tai, Semi-Accordant, *or* Accordant. *$\mathbf{K}^{\mathcal{C}}|_{\mathcal{F}_N} : \mathcal{F}_N \times \mathcal{F}_N \to \mathbb{R}^+_0$ is positive semi-definite for an arbitrary N if and only if $\sigma : \Sigma \times \Sigma \to \mathbb{R}^+_0$ is positive semi-definite.*

Theorem 1 has a wide range of applications. For example, the subtree-congruent mapping class is absorbent and transitive. Moreover, let Leaf-Tai, Leaf-Semi-Accordant, and Leaf-Accordant respectively denote the subclasses of Tai, Semi-Accordant and Accordant such that, for M belonging to the subclasses, x and y are both leaves if $(x, y) \in M$. They are also absorbent and transitive. Therefore, the counting functions for those mapping classes are positive semi-definite.

5 Conclusion

A generalization of the tree kernels due to Collins and Duffy [2], and due to Kashima and Koyanagi [3] is addressed in this paper. Based on the notion of tree mapping, which depicts a common sub-pattern between two trees, it is shown that these existing kernels are the counting functions of tree mappings. By focusing on four major classes of tree mappings proposed in the field of the tree edit distance, four counting functions of tree mappings are proposed according to the four classes. In addition, it is proved that three of the four counting functions are kernel functions, and the other is not by checking their positive semi-definiteness. One of the three tree kernels developed in this paper turns out to be the elastic tree kernel due to Kashima and Koyanagi. The other two tree kernels are more general than existing tree kernels in the interpretation of the common patterns occurring between two trees.

For future work, it is necessary to undertake an empirical analysis of the proposed tree kernels by applying them to real-world data in order to show the adequacy and effectiveness for specific applications.

Acknowledgments

We would like to thank the reviewers for their helpful comments. This work is partly supported by Grant-in-Aid for Scientific Research No. 17700138 from the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

1. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press (2004)
2. Collins, M., Duffy, N.: Convolution Kernels for Natural Language. Advances in Neural Information Processing Systems 14, Vol.1 (NIPS). (2001) 625–632
3. Kashima, H., Koyanagi, T.: Kernels for Semi-Structured Data. Proc. of the 9th International Conference on Machine Learning (ICML). (2002) 291–298
4. Tai, K.C.: The Tree-to-Tree Correction Problem. JACM **26** (1979) 422–433
5. Zhang, K.: Algorithms for The Constrained Editing Distance between Ordered Labeled Trees and Related Problems. Pattern Recognition **28** (1995) 463–474
6. Lu, C.L., Su, Z.Y., Tang, G.Y.: A New Measure of Edit Distance between Labeled Trees. LNCS 2108 (2001) 338–348
7. Richter, T.: A New Measure of The Distance between Ordered Trees and its Applications. Technical Report 85166-CS, Dept. of Computer Science, Univ. of Bonn (1997)
8. Jiang, T., Wang, L., Zhang, K.: Alignment of Trees — An Alternative to Tree Edit. Theoretical Computer Science **143** (1995) 137–148
9. Wang, J.L., Zhang, K.: Finding Similar Consensus between Trees: An Algorithm and A Distance Hierarchy. Pattern Recognition **34** (2001) 127–137
10. Kuboyama, T., Shin, K., Miyahara, T., Yasuda, H.: A Theoretical Analysis of Alignment and Edit Problems for Trees. Theoretical Computer Science, The 9th Italian Conference. LNCS 3701 (2005) 323–337
11. Shin, K., Kuboyama, T.: An algebraic Foundation of The Tree Edit Distance. Technical report, Center for Collaborative Research, University of Tokyo (2005)
12. Haussler, D.: Convolution kernels on discrete structures. UCSC-CRL 99-10, Dept. of Computer Science, University of California at Santa Cruz (1999)

Mining Interpretable Subgraphs

Siegfried Nijssen

Institut für Informatik, Albert-Ludwidgs-Universität,
Georges-Köhler-Allee, Gebäude 097, D-79110, Freiburg im Breisgau, Germany.
snijssen@informatik.uni-freiburg.de

Abstract. We present a measure that estimates the interpretability of a frequent subgraph. We show that a feature selection algorithm that uses this measure creates a set of features that is smaller and equally predictive as features obtained in earlier studies. A significant number of the selected features turn out to be trees or cyclic graphs, leading us to the conclusion that such features are not as useless as suggested in some earlier studies. Finally, we show that a constraint on this measure can be pushed in the mining process, thus leading to faster discovery of interesting subgraphs.

1 Introduction

One of the most important applications of graph mining is the analysis of molecular datasets, where the aim is to predict accurately if an unseen molecule exhibits a certain chemical activity or not. Traditional methodes proposed in the chemoinformatics community rely mostly on paths, but the data mining community has started the use of subgraphs in the hope of obtaining better classifiers, where both frequent pattern mining [10, 7] and kernel approaches [6, 9] have been investigated.

In two recent papers, however, it was suggested that the additional complexity of taking into account subgraphs may not be justified [2, 9]. It was reported that for a large number of classifiers and a large number of datasets, features based on paths yield more accurate classifiers.

In the local pattern mining approach, which is used for example in [2], classification is typically approached as follows. First, given a dataset, a set of patterns (in this case, paths or subgraphs) is computed. During this search, a constraint is enforced such that only patterns are found that have sufficient correlation with the activity of the molecules; for instance, the subgraph should occur more often in the active molecules than in the inactive ones.

Second, each pattern is used to create a new feature for the molecules: if a molecule contains a certain pattern, the feature correponding to the pattern gets value 1 (or true), otherwise the feature has value 0 (or false). On the resulting set of features, a classifier, such as a decision tree, is learned. This decision tree is finally used to predict the activity of new molecules.

It is indeed an interesting observation that a set of paths can contain the same amount or even more information than a set of subgraphs. An equally

interesting question is how these results can be explained. We believe that there are several possible explanations:

- in [2] feature sets of equal size were determined for each type of pattern; if one considers the contingency tables that achieve the best correlation values, there are usually significantly more subgraphs with exactly that table than there are subpaths, yielding features that are more correlated among each other in case a fixed number of subgraphs are used;
- if subgraphs are considered, the amount of possible features is much larger; thus the risk of overfitting to the training data is also much larger.

In the study of [2] it was reported that by relaxing the correlation constraint, a more accurate classifier can be obtained. One could explain this by the argument that the resulting additional features are in this case less correlated among eachother.

In this paper, we investigate these observations further. Concretely, we propose a measure to drastically cut down the number of frequent subgraphs that are used as features, thus yielding classifiers that are more interpretable, and we propose a feature selection method to avoid redundancy among the features. We show that the interpretability constraint can be pushed in the mining process, thus showing that there are efficient ways to search for patterns that are still simple, but also cyclic.

The paper is organized as follows. In section 2 we briefly reintroduce the formal concepts that are important in this paper. In section 3 we introduce our notion of interpretable subgraphs. In section 4 we show how a constraint on interpretable subgraphs can be pushed in the mining process. Section 5 introduces our feature selection algorithm. Section 6 concludes.

2 Concepts

An essential task in the local pattern mining scenario is the computation of a relevant set of patterns. In the data mining community, these features are often determined using a *frequent subgraph mining* algorithm. To formalize what a frequent subgraph is, we need the following definitions.

Definition 1 (Graphs). *An undirected, labeled graph* $G(V, E, \lambda, \Sigma)$ *consists of a finite set* V *of vertices, a set* $E \subseteq \{\{u, v\} | u, v \in V, u \neq v\}$ *of edges, an alphabet* Σ *and a labeling function* $\lambda : (V \cup E) \rightarrow \Sigma$.

Definition 2 (Connected Graphs). *A graph* $G(V, E, \lambda, \Sigma)$ *is called connected iff for each pair of nodes* $u, v \in V$ *there is a sequence of nodes* $v_1, \ldots, v_n$ *in* V *with* $v_1 = u$ *and* $v_n = v$ *and* $\{v_i, v_{i+1}\} \in E$.

In this paper, we only consider connected graphs.

Definition 3 (Graph Isomorphism). *Graphs* $G(V, E, \lambda, \Sigma)$ *and* $G'(V', E', \lambda', \Sigma')$ *are called isomorphic if there exists a bijective function* $f : V \rightarrow V'$ *such that:* $\forall v \in V : \lambda(v) = \lambda'(f(v)) \wedge E = \{\{f(v_1), f(v_2)\} | \{v_1, v_2\} \in E'\} \wedge \forall e \in E : \lambda(e) = \lambda'(f(e))$.

Definition 4 (Subgraph). *Given a graph* $G(V, E, \lambda, \Sigma)$*,* $G'(V', E', \lambda', \Sigma')$ *is called a subgraph of* G *iff* $V' \subseteq V \wedge E' \subseteq E \wedge \forall v \in V' : \lambda(v') = \lambda(v) \wedge \forall e \in E' : \lambda(e') = \lambda(e)$.

If a graph H has a subgraph H' such that a graph G is isomorphic with H', then G is said to be *subgraph isomorphic* with H, denoted by $H \succeq G$; fuction f defines how subgraph G can be *embedded* in the larger graph H. The set of all embeddings of G in H is denoted by $F_{H \succeq G}$.

Definition 5 (Frequent Subgraph). *Given a collection* $\mathcal{D}$ *of graphs and a graph* G*, the* frequency *of* G*, denoted by* $freq(G, \mathcal{D})$*, is the cardinality of the set* $\{G' \in \mathcal{D} | G' \succeq G\}$*. A graph* G *is* frequent *if* $freq(G, \mathcal{D}) \geq minsup$*, for a predefined threshold* $minsup$*.*

The problem that is solved by a *frequent subgraph mining* algorithm is to find *all* subgraphs that are frequent in a dataset. In the molecular setting the set of frequent subgraphs is usually computed on the active set of molecules, and their corresponding frequencies on the inactive set of molecules are computed later.

Several algorithms have been proposed to solve the frequent subgraph mining problem, among which gSpan [10], Gaston [7] and MoFA [5, 8]. All these algorithms exploit the fact that the frequency constraint is anti-monotonic: if $G \succeq H$, then $freq(G) \leq freq(H)$. As a consequence of this property, algorithms do not need to consider all possible subgraphs of a database to find all frequent ones: it is possible to search from small to large subgraphs, and stop searching if a large graph is not frequent any more.

An important issue that subgraph miners have to address, is that there are many ways to construct isomorphic subgraphs. For instance, a molecular fragment consisting of an oxygen (O) connected to a carbon (C), can be obtained by connecting either a carbon to an oxygen, or by connecting an oxygen to a carbon. To avoid duplicates, only one of these two extensions should be allowed. Most subgraph miners solve this issue by using a *canonical code* that restricts in what ways a subgraph can be extended.

The *degree* of a node v, denoted by $deg(v)$, is the number of nodes to which the node is directly connected by the edges in the graph. Several simple classes of subgraphs have been studied. A *(free) tree* is a connected graph in which $|V| = |E| + 1$; a *path* is a tree in which no node has a degree higher than 2.

Assume that there are two graphs H and G, such that $H \succeq G$, and $freq(G) = freq(H)$, then for the purpose of classification graph H can be considered redundant: it covers exactly the same set of molecules as G, but is more specific, and therefore more likely to overfit. Therefore, it is often useful to restrict the search to subgraphs G for which there is no subgraph H for which $freq(G) = freq(H)$. Such subgraphs are called *free subgraphs*. If there is no supergraph H for which $freq(G) = freq(H)$, then subgraph G is called a *closed subgraph*.

Finally, if we know the frequency of a subgraph in both a set of active and inactive molecules, we can compute its *correlation* with that activity; this score —typically the result of a χ^2 test— can be used to rank patterns. A *top-k* pattern miner finds all k subgraphs that obtain the highest positions in this

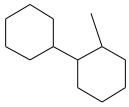
ranking. The motivation is that only the highest scoring subgraphs are expected to be important to classify molecules.

In [2], the classification of molecules was approached using a *top-k free* pattern mining algorithm. For several values of k the highest scoring patterns were determined; these patterns were used to build classifiers. Trees, graphs and paths were considered as pattern domains. It was concluded in many cases that classifiers based on paths achieved the highest predictive accuracy.

3 Interpretable subgraphs

A situation that occurs in many molecular data mining applications, is illustrated by the following example. Assume that we have a (hypothetical) database containing the following two highly active molecules:

Then we can derive the following substructure, which tries to encode a disjunction of these two graphs, but which is chemically hard to interpret, as it contains only part of an aromatic ring:



In this subgraph, some edges have been selectively included or excluded to obtain the best fit to the data. This subgraph shows that there is sometimes a tension between *interpretability* and predictive *accuracy* of subgraphs:

- if this subgraph obtains a superior accuracy score, one could believe that it is desirable that it is found;
- if this subgraph cannot be interpreted without looking through an enormous set of embeddings in the data, it may not be useful that it is found.

Thus, by *interpretability* we mean that less additional information has to be considered before the true value of the pattern can be determined by an expert.

A further problem is the size of the search space. If one allows individual removal or inclusion of edges, the search space becomes larger. As a result of the additional features, accuracy can increase, but on the other hand, if there are so many features to consider, it will be harder to interpret the features and to determine which of the features are really of interest.

These issues also important when building accurate classifiers. On the one hand, tweaking of substructures can be desirable, as it could yield a better fit to the data; on the one hand, if too much tweaking is allowed, we might *over*fit to the data. It is a well-known problem to balance these issues.

Until now, two solutions are prevalent. In the approaches based on gSpan or Gaston [10, 7] the interpretability of subgraphs is neglected, thus allowing

structural tweaking as in the example above. In MoFA [5, 8] the issue is addressed by hard-coding structural preference in the search: for example, rings of length 5 or 6 are marked as special structural elements, and edges of such rings can never be added individually, yielding more interpretable subgraphs and smaller search spaces, but disallowing potentially interesting subgraphs such as in the example above.

In this paper, we investigate an approach in which there is a parameter to balance interpretability and structural freedom. We wish to allow structural freedom, but want to implement a measure that allows us to express our preference for structures that are not too incomplete. There are many conceivable measures. Here, we take an approach that is based on considering how graphs match to the data. What we can observe in general for the example fragment, is that it is *incomplete*: in every embedding of this subgraph, we can connect an additional edge to the fragment. If we consider the original molecules as fragments, they are more 'complete': there would usually be less possibilities to connect additional edges to an embedding. Therefore, a reasonable measure seems to be based on the number of edges that can be connected to an embedding of the subgraph.

Formally, we can compute this as follows:

$$missing(f) = \sum_{v \in V_G} deg(f(v)) - deg(v),$$

which is the number of edges that can be connected to the embedded subgraph. For a database $\mathcal{D}$ of graphs we can compute

$$missing(G) = \frac{\sum_{H \in \mathcal{D}, G \subseteq H} \min_{f \in \mathcal{F}_{H \succeq G}} missing(f)}{|\{H \in \mathcal{D} | H \succeq G\}|}.$$

We believe that it is reasonable to consider only the best possible embedding (for example, an average over all embeddings would be skewed as the number of embeddings can be exponential), therefore we choose to minimize the *missing* value. The measure clearly reflects the number of possibilities for extending the pattern in the minimal case.

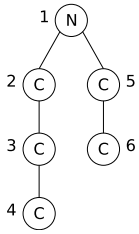
The use of this measure can be two-fold. We can use it to express a preference between structures that are more or less equivalent otherwise, thus in principle still allowing unlimited structural freedom. Another possibility is to enforce a constraint on this measure. In this case, we might not find some well-scoring substructures any more. On the other hand, structures satisfying such a constraint would have the advantage that the resulting substructures are usually easier to interpret, and the possibilities for overfitting are reduced.

It is important to observe that this constraint is neither monotonic nor anti-monotonic. A given subgraph with a low *missing* value can obtain a high *missing* value after extenion: assume that the newly added node is always connected to many other nodes in the data, then *missing* will go up. On the other hand, the *missing* value can also go down. If we add an edge to the example fragment, we can obtain the second example molecule. This graph has less missing edges.

4 Pushing the *missing* constraint

Although the $missing(G) \leq maxmissing$ constraint is neither monotonic nor anti-monotonic, it is still possible to push the constraint in the mining process. The reason is that in subgraph mining, the canonical form limits how we are allowed to refine a subgraph. It is not possible to connect a new edge to every possible node in a subgraph to avoid duplicates. Therefore, for a certain embedding, we can determine that we will never get rid of some of the 'missing' edges as the search procedure does not allow for adding these missing edges. If the number of mising edges that we are not allowed to get rid off, is too large, we can stop refining the subgraph and prune the search space.

As an example, we use the DFS canonical code of gSpan; for more details about this canonical code, consult [10]. Consider the subgraph given below.



We assume that the label N is lower than the label C. Then in gSpan the canonical DFS code of this graph is (assuming all edges are labeled with λ):

$$(1, 2, N, \lambda, C)(2, 3, C, \lambda, C)(3, 4, C, \lambda, C)(1, 5, N, \lambda, C)(5, 6, C, \lambda, C),$$

where $(x, y, \sigma_1, \sigma_2, \sigma_3)$ denotes that there is an edge from node v_x to node v_y with label σ_2; the label of v_x and v_y is σ_1 and σ_3, respectively. According to gSpan's canonical code, we can only connect new edges to the *rightmost* path. Thus, nodes 2, 3 and 4 cannot be extended any more. Furthermore, in the DFS code, among siblings, the lowest label must be listed first. This means that we cannot connect a node with label N to node 1 of the example graph.

For a given node v in a subgraph, and an embedding f of this subgraph, let $sdeg_f(v)$ be the *static degree* of the node. We define the static degree to be the number of sibling edges of $f(v)$ that can not be added to the subgraph, either because they are already in the subgraph, or because the canonical code does not allow it. Then for an embedding we can compute

$$smissing(f) = \sum_{v \in V_G} sdeg_f(v) - deg(v).$$

For a given graph H in the database and a pattern graph G, we can determine whether

$$\min_{f \in \mathcal{F}_{H \geq G}} smissing(f) \leq maxmissing.$$

If there is no database graph in which this condition holds, we do not need to refine the subgraph, as the average *missing* value can never become low enough to satisfy the constraint.

Name	Size class 1	Size class 2
NCI HIV	417	1069
Mutagenicity I [7]	2401	1936
Mutagenicity II [4]	341	343
Biodegradability [1]	143	185
PTE [10]	340	—

Table 1. Properties of the datasets used in the experiments

In principle, to decide which edges can be connected to a subgraph according to the canonical form is computationally hard, as we need to solve graph isomorphism. However, the above given two rules can be used to approximate $sdeg_f(v)$: nodes are static if either (1) they are a sibling of a node not on the rightmost path or (2) they are a siblings of a node on the rightmost path, but the label is too low. This approximation will not influence the correctness of the algorithm, as it underestimates the number of missing edges.

Experimental Evaluation To test the influence of this pruning rule we have performed several experiments with an implementation of gSpan that includes this constraint. The properties of the datasets are listed in Figure 1.

We first performed experiments on the predictive toxicology dataset (PTE). Runtime experiments were performed on an Intel Pentium M processor running at 1.1Ghz, and are listed in Table 2. In this table, the number of processed subgraphs is the number of subgraphs that is enumerated by the graph mining algorithm, including subgraphs for which *missing* is too high. We can observe

min-sup	*max-missing*	runtime	# processed subgraphs
32	∞	5.9s	930
32	2.0	3.7s	706
16	∞	46.9s	4405
16	2.0	9.7s	2197
10	∞	261.8s	22758
10	4.0	108.9s	13472
10	3.0	57.0s	9449
10	2.0	19.9s	5344

Table 2. Experiments on the PTE dataset

min-sup	*max-missing*	runtime	# processed subgraphs
200	∞	298.7s	2244
200	3.0	298.9s	2231
200	2.0	279.5s	2115
150	∞	590.5s	6367
150	2.0	533.9s	5790

Table 3. Experiments on the Mutagenesis I dataset

that the runtime improvement is much larger than the number of processed subgraphs. The reason is that especially large subgraphs, for which computing subgraph isomorphism is more expensive, are pruned.

Next, we performed a set of experiments on the Mutagenesis I dataset. Experiments were performed on an Intel Pentium IV running at 3.2Ghz. The minimum support threshold applies to the set of active molecules. We observe that on this

dataset the influence of the constraint is almost negligible. A possible explanation is that in this dataset the molecules were encoded without hydrogens (contrary to the PTE dataset). Consequently, the branching factor of nodes in the data is lower. A larger amount of the nodes therefore already have low *missing* value. The pruning opportunities are therefore reduced.

Finally, we performed an experiment in which the NCI HIV dataset was used; we determined the frequent patterns in the active molecules, and evaluated the patterns also on the moderately active ones. The experiments were performed on an Intel Pentium IV 3.2Ghz again. For a minimum support of 6% we determined that the runtime without constraint was 1500s, while it was only 691s if a *maxmissing* constraint of 3.0 was applied. In this dataset the hydrogens were not included in the encoding. Apperently, the amount of speed-up is not only dependent on this encoding.

Dataset	*minsup*	Frequent	Interpretable
Mutagenesis I	2.5%	48583	132
Mutagenesis II	2.5%	7577	208
Mutagenesis II	5.0%	1005	103
HIV	6.0%	371374	350
HIV	7.5%	58380	264
Biodegradability	2.5%	168033	206
Biodegradability	5.0%	34839	88

Table 4. Experiments showing the reduction in subgraphs by applying the *maxmissing* constraint

Finally, in Table 4 the results are reported of an experiment in which we compared the number of frequent subgraphs in the output of gSpan with and without a $maxmissing = 3.0$ constraint. This shows that the *maxmissing* constraint is very effective in reducing the number of subgraphs.

5 Ranking patterns

As discussed in the introduction, the simplest approach to rank subgraphs is to order them according to the χ^2 statistic of their correlation with the activity of molecules. However, such an absolute ordering may not always be justified. Assume that we have two subgraphs of which the TID lists (i.e., the lists of graphs in the database to which they map) are *almost* equal, then it might not be justified to prefer the one pattern above the other pattern; the small difference might just be noise in the data. For every pattern, we can therefore define a set of patterns that is close to it in terms of TID list, and for which we have no reason to prefer one pattern above the other.

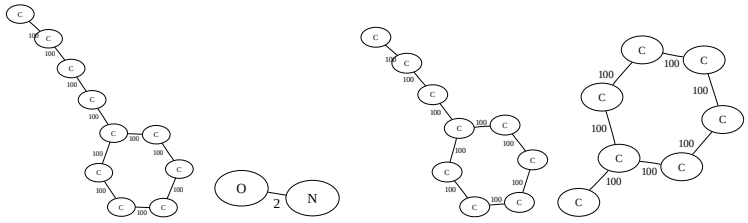
One way then to cut the ties between patterns with almost equal TID lists, is to use the *missing* score. Thus, assuming that we have sorted all patterns on χ^2, we can apply this procedure:

1. Among the set of all patterns within a close distance from the topmost pattern, pick the pattern that scores best according to the *missing* score. Add this pattern to the result set.
2. Remove all patterns that are close to the chosen pattern (which includes the best pattern), including the chosen pattern.
3. Go to step 1 for the remaining sorted list of patterns.

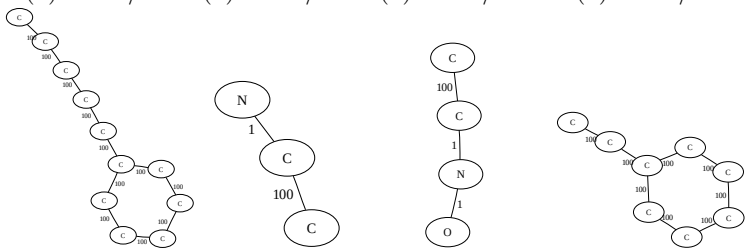
Observe that if two graphs have equal TID lists, and one is a subgraph of the other, the *missing* score can still cut ties. However, one cannot predict whether the subgraph or the supergraph will be preferred. Thus, we do not imply either closed subgraph mining or free subgraph mining.

Parameters in this procedure are the distance measure and the threshold on this measure. Currently, we have chosen symmetric difference as distance measure. As a rule of thumb, the threshold on the distance is lower than the minimum support threshold.

We first performed an experiment on the Mutagenesis I dataset. We used a minimum support of 150, a distance threshold of 30, and no interpretability constraint. The total number of frequent patterns was 6367. If we apply the selection procedure given above, 297 fragments remain. The best 8 fragments are given below. For each fragment its χ^2 value and its *missing* value are given.

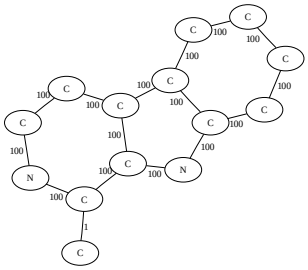


(1) 401.0/4.3 (2) 384.8/1.8 (3) 364.9/4.0 (4) 352.8/3.1



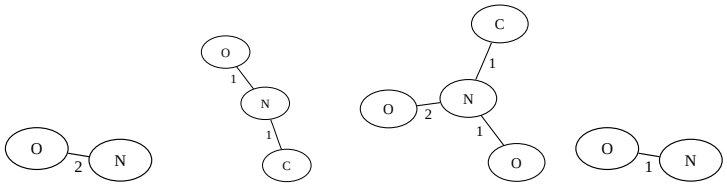
(5) 348.5/4.6 (6) 337.4/3.2 (7) 334.4/3.5 (8) 327.5/5.3

We can observe that there is a large number of fragments involving aromatic bonds, which are labeled with '100' in these figures. The first, third and fourth fragment still have significant differences in TID lists. Please note that the long chains in the patterns are *not* chains in the data. The method of [8] would not prune them. An example of a molecule that does not contain the first fragment, but does contain the third fragment, is given below:

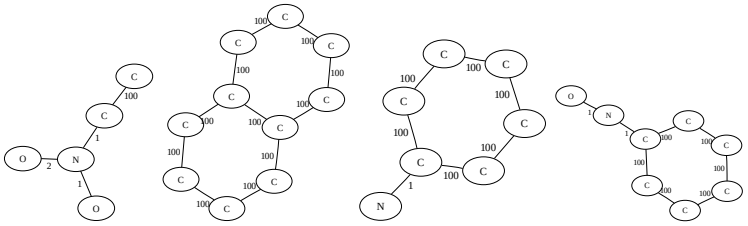


The problem is that the information of how this pattern matches (i.e., if the non-branching chain in the pattern is also a non-branching chain in the data, as one could be tempted to think), is not readily available from the pattern and a closer inspectation of patterns and data is required; there might also be other explanations. It is not obvious for what reasons this particular fragment matches as good as it does.

If however we apply a maximum value of 3.0 as constraint on the missing value, the number of frequent fragments that satisfies the constraint falls down to 54. With the algorithm given above, using a threshold distance of 30, 44 become selected. The best 8 fragments are given below:



(1) 384.8/1.8 (2) 322.8/2.9 (3) 303.2/2.0 (4) 312.1/1.9



(5) 304.9/2.0 (6) 288.4/2.7 (7) 238.9/2.9 (8) 227.9/3.0

To a certain extent, we still have similar effects as in the previous experiment. The fifth fragment only contains one aromatic bond, because the given group can connect to any aromatic ring, not only a six-ring, like in the seventh fragment. The effects are however much smaller.

Next, we performed several experiments on this dataset to compare the classification accuracies of several classes of fragments. To make the results comparable to the results obtained in [2], we took the same settings as in that paper, except that we used our new algorithm to select the cyclic subgraphs that are used as features; so, the results are obtained after 10 fold crossvalidation, using implementations of Weka [3]. We restricted ourselves to the C4.5 decision tree

Algorithm	Mutagenesis I		Mutagenesis II		Biodegradability	
	Number of features	Accuracy	Number of features	Accuracy	Number of features	Accuracy
Paths	100	76.37	100	70.90	100	76.22
	1000	79.73	1000	71.63	1000	74.10
Trees	100	70.94	100	70.39	100	71.96
	1000	74.83	1000	71.55	1000	76.07
Graphs	100	70.79	100	70.53	100	71.82
	1000	74.68	1000	72.06	1000	71.20
Selection 2.5%	82	79.94	93	76.02	83	72.56
Selection 5.0%	48	78.35	58	70.91	54	72.87

Table 5. Accuracies on the Mutagenesis I, Mutagenesis II and Biodegradability datasets; Selection $x\%$ denotes that a minimum support of $x\%$ was used

Algorithm	Number of features	Accuracy	Algorithm	Number of features	Accuracy
Paths	100	77.46	Selection 6.0% 4	107	82.77
	1000	83.21	Selection 6.0% 8	93	82.30
Trees	100	77.06	Selection 7.5% 4	94	82.77
	1000	75.95	Selection 7.5% 8	83	82.55
Graphs	100	77.06			
	1000	75.95			

Table 6. Accuracies on the NCI HIV dataset; Selection $x\%$ y denotes that a minimum support of $x\%$ was used and a maximum distance of y

learner, as this algorithm achieved the best results in all experiments in [2]. Results are listed in Table 5 and 6; results for Paths, Trees and Graphs are copied from [2].

In all cases *maxmissing* was fixed to 3.0. The distance threshold was 60 on Mutagenesis I, 2.5% on Mutagenesis II, and 2.5% on Biodegradability. Interestingly, we achieve similar classification accuracies as for the paths in most cases; only on the biodegradability dataset our results are disappointing. The set of features that is passed to C4.5 is smaller in all cases. Also in terms of the size of the decision tree, our classification method seems more interpretable; for example, a decision tree learned on the entire Mutagenesis I dataset using 1000 paths, contains 196 leaves, while the tree that is built on our features contains 104 leaves.

Comparing these tables to table 4, the largest reduction in the number of features stems from the *maxmising* constraint.

In addition to the minimum support threshold, we also varied the distance threshold on the NCI HIV dataset. Most results do not seem to be very sensitive to the choice of the parameters, with the exception perhaps of the *minsup* threshold on the Mutagenesis II dataset.

6 Conclusions

We argued that there is a general trade-off between pattern representations, mining efficiencies and classification accuracies. To illustrate this, we developed a measure to quantify the interpretability of a subgraph, and showed that this measure could be pushed in the mining process, with mixed success. We performed experiments with the features that were selected using a feature selection method that takes this measure into account, and found that the classifiers were accurate in most cases, and possibly more interpretable.

There are many open issues in this approach. A broader study of interpretability measures might yield measures that approximate the average chemist's idea of interpretability better. For instance, one could also base an interpretability measure on how well examples cluster on the frequent supergraphs of a subgraph. We used an ad-hoc method to prefer structures that achieve a higher interpretability score, but a more principled approach would be desirable. Finally, in most experiments we applied feature selection only on pruned features. Our feature selection method did not work efficiently on large amounts of features. How to optimize this method is also an open question.

Acknowledgements This work was supported by the EU FET IST project IQ ("Inductive Querying"), contract number FP6-516169. We would like to thank Taneli Mielikäinen and Jeroen Kazius for discussions.

References

1. H. Blockeel, S. Dzeroski, B. Kompare, S. Kramer, B. Pfahringer, and W. Van Laer. Experiments in predicting biodegradability. In *Appl. Art. Int. 18*, pages 157–181, 2004.
2. B. Bringmann, A. Zimmermann, L. De Raedt, and S. Nijssen. Don't be afraid of simpler patterns. In *PKDD*, 2006.
3. E. Frank, M. Hall, L.E. Trigg, G. Holmes, and I.H. Witten. Data mining in bioinformatics using weka. In *Bioinformatics 20*, pages 2479–2481, 2004.
4. C. Helma, T. Cramer, S. Kramer, and L. De Raedt. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. In *Journal of Chemical Information and Computer Systems 44*, pages 1402–1411, 2004.
5. H. Hofer, C. Borgelt, and M. Berthold. Large scale mining of molecular fragments with wildcards. In *IDA*, pages 380–389, 2003.
6. T. Horvath, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *KDD*, pages 158–167, 2004.
7. J. Kazius, S. Nijssen, J.N. Kok, T. Bäck, and A. IJzerman. Substructure mining using elaborate chemical representation. In *Journal of Chemical Information and Modeling 46*, 2006.
8. T. Meinl, C. Borgelt, and M. Berthold. Mining fragments with fuzzy chains in molecular databases. In *MGTS*, pages 49–60, 2004.
9. N. Wale and G. Karypis. Acyclic subgraph-based descriptor spaces for chemical compound retrieval and classification. In *Technical report, Univ. Minnesota*, 2006.
10. X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.

A Linear Programming Approach for Molecular QSAR analysis

Hiroto Saigo[1], Tadashi Kadowaki[2], and Koji Tsuda[1]

[1] Max Planck Institute for Biological Cybernetics,
Spemannstr. 38, 72076 Tübingen, Germany
{hiroto.saigo,koji.tsuda}@tuebingen.mpg.de,
[2] Bioinformatics Center, ICR, Kyoto University
Uji, Kyoto 611-0011, Japan
tadakado@gmail.com

Abstract. Small molecules in chemistry can be represented as graphs. In a quantitative structure-activity relationship (QSAR) analysis, the central task is to find a regression function that predicts the activity of the molecule in high accuracy. Setting a QSAR as a primal target, we propose a new linear programming approach to the graph-based regression problem. Our method extends the graph classification algorithm by Kudo et al. (NIPS 2004), which is a combination of boosting and graph mining. Instead of sequential multiplicative updates, we employ the linear programming boosting (LP) for regression. The LP approach allows to include inequality constraints for the parameter vector, which turns out to be particularly useful in QSAR tasks where activity values are sometimes unavailable. Furthermore, the efficiency is improved significantly by employing multiple pricing.

1 Introduction

Nowadays we are facing a problem of screening a huge number of molecules in order to testify, e.g., if it is toxic to human or it has an effect on HIV virus, etc. Such bioactivities or chemical reactivities are measured by laborious experiments, so selecting small number of good candidates for the later synthesis is important. A quantitative structure-activity relationship (QSAR) analysis, a process to relate a series of molecular features with biological activities or chemical reactivities, is expected to decrease a number of expensive experiments. The conventional QSAR analysis manipulates chemical data in a table in which molecules are defined by individual rows and molecular properties (descriptors) in binary or real values are defined by the associated columns. The prediction model is then built to be consistent to the structure-activity relationship. Our approach use subgraphs as molecular properties instead of conventional descriptors. Since we know that simple subgraphs are already included in conventional descriptors, and believe that enriching subgraph features would contribute to make a better prediction model.

A graph is a powerful mathematical framework which can deal with many real-world objects. Several approaches based on kernel methods have been proposed [1–6], which all consider molecules as graphs and tried defining distance measures between molecules. However, these kernel methods lack interpretability, because the feature space is implicitly defined and it is difficult to figure out which features played an important role in prediction.

We take the boosting approach, which works in a feature space explicitly defined by substructure indicators [7]. Therefore, it is rather easy to show the substructures contributed to activity predictions, which may lead to new findings by chemists. Though the number of possible subgraphs in graph database are exponentially large, the recent advance of graph mining algorithms [8–12] suggests a way to handle them. The graph boosting method by Kudo et al. [7] has to be modified in several ways for QSAR applications. First of all, the original classification algorithm should be modified to a regression algorithm, because the activity is real-valued. Very recently, Kadowaki et al.[13] used the graph boosting in a SAR task, where the problem is to predict a chemical compound is active or not. However, the activity values are continuous and they are not obviously separated into active/non-active categories. Second, in publicly available databases, the activity values are not always available, because, if some compounds are obviously inactive, they do not bother to measure the activity. Therefore, for many compounds, one knows that their activities are low, but the actual values are not available. We need a mechanism to take those unusual data into account. Finally, in AdaBoost, only one substructure is found by the search of the whole pattern space. So, if one needs d substructures for good accuracy, the graph mining has to be done d times. For more efficiency, it is desirable that multiple substructures are derived by one mining call.

Among several boosting algorithms for regression [14], we found the linear programming (LP) boosting proposed by Demiriz et al. [15] is most appropriate for our task. One reason is that the linear programming allows to include inequality constraints which are useful for incorporating the compounds with low activities. Another reason is that it is possible to obtain multiple structures by one graph mining call, because the LP boost always updates all the parameters whereas AdaBoost updates one parameter at a time. Finally, in the LP boost, the optimality of the solution can be evaluated by the duality gap, whereas, in AdaBoost, it is not obvious when to stop the iteration and it is hard to figure out the distance from the current solution to the optimal one.

In this paper, we will describe how the LP boost can be combined with the graph mining algorithm to yield an efficient graph regression algorithm. In experiments using Endocrine Disruptors Knowledge Base (EDKB), our method is favorably compared with the marginalized graph kernels [1] that are successfully applied to chemical data recently [5]. We also illustrate the speed up achieved by reducing the number of mining calls.

2 Graph Preliminaries

In this paper, we deal with undirected, labeled and connected graphs. To be more precise, we define the graph and its subgraph as follows:

Definition 1 (Labeled Connected Graph). *A labeled graph is represented in a 4-tuple $G = (V, E, \mathcal{L}, l)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, $\mathcal{L} \in \mathcal{R}$ is a set of labels, and $l : V \cup E \rightarrow \mathcal{L}$ is a mapping that assigns labels to the vertices and edges. A labeled connected graph is a labeled graph such that there is a path between any pair of vertices.*

Definition 2 (Subgraph). *Let $G' = (V', E', \mathcal{L}', l')$ and $G = (V, E, \mathcal{L}, l)$ be labeled connected graphs. G' is a subgraph of G ($G' \subseteq G$) if the following conditions are satisfied: (1)$V' \subseteq V$, (2)$E' \subseteq E$, (3)$\mathcal{L}' \subseteq \mathcal{L}$, and (4)$l' \subseteq l$. If G' is a subgraph of G, then G is a supergraph of G'.*

To apply a machine learning method to graphs, one has to represent a graph as a feature vector. One idea is to represent a graph of a set of paths as in marginalized graph kernels (MGK) [1]. MGK and similar methods are recently applied to the classification of chemical compounds [4, 5]. Although the computation of kernels (i.e., the dot product of feature vectors) can be done in polynomial time using the path representations, paths cannot represent structural features such as loops and often end up with poor results (e.g., [16]). Therefore, we employ the substructure representation (Figure 1), where the feature vector consists of binary indicators of *patterns* (i.e., small graphs). Now our central issue is how to select patterns informative for regression. We will adopt the boosting approach to solve the problem as described in the next section. In chemoinformatics, it is common that a set of small graphs (i.e., fingerprints) is determined a priori, and the feature vector is constructed based on them [17]. However, we do not rely on ready-made fingerprints to search for unexplored features and to make our method applicable to any graph regression problem in other areas of science.

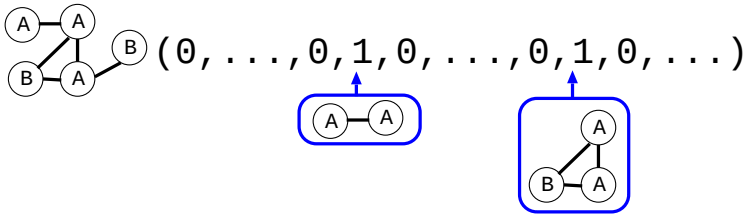


Fig. 1. Substructure representation of a graph. Each substructure is restored in the corresponding position in a vector.

3 Graph Regression by Linear Programming

When a graph is represented as a vector of binary indicators of all possible substructures, the dimensionality becomes too large for conventional regression methods such as ridge regression. In this work, we employ a regression method based on the LP (linear programming) boosting [15, 18], because it greedily selects features in learning and can avoid computational problems in a systematic way. In feature selection, we need to search for the best feature in the whole pattern space. To perform the search efficiently, we adopted a data structure called DFS code tree [12] as will be described in the next section. A QSAR problem is basically considered as a graph regression problem, where graph-activity pairs are given as the training data, and the activities of test graphs are predicted by the learned regression function. However, one problem is that the activity is measured only for the chemicals that are suspected to be active. For apparently inactive chemicals, nobody bothers to measure their activities. Thus, in the database, we have a set of chemicals with real-valued activities and a set of chemicals known to be inactive but the actual activity values are not available. The latter examples are called "clearly negative data". In the following formulation, those examples appear as inequality constraints of the weight vector.

Let $\mathbf{x} \in \mathcal{R}^d$ be a feature vector. Given the training examples $\{\mathbf{x}_i, y_i\}_{i=1}^m$ and clearly negative examples $\{\bar{\mathbf{x}}_k\}_{k=1}^l$, our objective is to learn the regression function

$$f(\mathbf{x}) = \sum_{j=1}^{d} \alpha_j x_j.$$

where α_j is a weight parameter. The objective function to minimize is as follows:

$$\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m} |\boldsymbol{\alpha}^\top \mathbf{x}_i - y_i|_\epsilon + \sum_{k=1}^{l} |\boldsymbol{\alpha}^\top \bar{\mathbf{x}}_k - z|_+ + \frac{1}{C} \sum_{j=1}^{n} |\alpha_j|$$

where z is a predetermined negative constant, C is the regularization parameter and $|\cdot|_\epsilon$ is the ϵ-insensitive loss [19], and $|\cdots|_+$ is the hinge loss, namely $|t|_+ = t\ (t \geq 0),\ 0\ (t < 0)$. We used the L1-regularizer to force most of the weights to be exactly zero in solution. Even if the dimensionality is large, the number of non-zero weights is kept small by this regularizer. The solution is obtained by solving the following linear programming.

$$\min_{\boldsymbol{\alpha},\boldsymbol{\xi}} \ ||\boldsymbol{\alpha}||_1 + C \sum_{i=1}^{m} \xi_i^+ + \xi_i^- + C \sum_{k=1}^{l} \xi'_k \tag{1}$$

$$s.t. \ \ \boldsymbol{\alpha}^\top \mathbf{x}_i - y_i \leq \epsilon + \xi_i^+, \quad \xi_i^+ \geq 0, \quad i = 1, \ldots, m \tag{2}$$

$$y_i - \boldsymbol{\alpha}^\top \mathbf{x}_i \leq \epsilon + \xi_i^-, \quad \xi_i^- \geq 0, \quad i = 1, \ldots, m \tag{3}$$

$$\boldsymbol{\alpha}^\top \bar{\mathbf{x}}_k \leq z + \xi'_k, \quad k = 1, \ldots, l \tag{4}$$

where ξ_i^+, ξ_i^- are slack variables for over-estimation and under-estimation, respectively, and ξ'_k is for clearly negative example. Let u_i^+, u_i^-, v_k be the Lagrange multipliers for the constraints (2), (3) and (4), respectively. Setting $u_i = u_i^+ - u_i^-$, the dual of the above problem is written as

$$\min_{\mathbf{u},\mathbf{v}} \; z\sum_{k=1}^{l} v_k - \sum_{i=1}^{m} y_i u_i + \epsilon \sum_{i=1}^{m} |u_i| \tag{5}$$

$$s.t. \; -1 \leq \sum_{i=1}^{m} u_i x_{ij} - \sum_{k=1}^{l} v_k \bar{x}_{kj} \leq 1, \; j = 1, \cdots, d \tag{6}$$

$$-C \leq u_i \leq C, \; i = 1, \ldots, m \tag{7}$$

$$0 \leq v_k \leq C, \; k = 1, \ldots, l \tag{8}$$

Instead of the primal problem, we will solve the dual problem and recover the solution for $\boldsymbol{\alpha}$ from the Lagrange multipliers of the dual problem [15, 18].

When the number of features d is extremely large, it is computationally prohibitive to solve the dual problem directly. Such a large scale problem is typically solved by the column generation (CG) algorithm [15, 18], where one starts from a restricted problem with a small number of constraints and necessary constraints are added one by one. At each step, the restricted LP problem is solved, and the solution at step t is used to select the constraint added in step $t+1$. The procedure continues until the convergence, or we can trade the accuracy with the computational time by stopping it before convergence. In our case, the problematic part is (6), so the column generation is performed with respect to the constraints in (6).

The efficiency of the CG algorithm depends crucially on the *pricing* step, where the importance of each constraint is evaluated using an intermediate solution. Here we select the constraint which is violated the most.

$$j^* = \arg\max_j \left| \sum_{i=1}^{m} u_i x_{ij} - \sum_{k=1}^{l} v_k \bar{x}_{kj} \right|. \tag{9}$$

If the maximum value is below or equal to 1, the column generation is stopped. It is also possible to add multiple constraints at a time. For example, one can sort the constraints based on the score (9), and take the top t constraints. This technique is called *multiple pricing* [20] and we will actually adopt it for reducing the number of searches.

When the substructure representation of a graph is employed, the number of all constraints is extremely large, thus we need a specialized machinery to obtain the maximally violated constraint. Since each constraint corresponds to a pattern, the search (9) is formulated as a graph mining problem as explained below.

4 Weighted Substructure Mining

Graph mining algorithms such as gspan efficiently enumerate the set of patterns that satisfy a predetermined condition [12]. Denote by $\mathcal{G} = \{G_i\}_{i=1}^n$ a graph database including l clearly negative examples ($n = m + l$), and let $\mathcal{T} = \{T_j\}_{j=1}^d$ be the set of all patterns, i.e., the set of all subgraphs included in at least one graph in $\mathcal{G}$. There are many variations of graph mining, but the most common one is the frequent substructure mining, where the task is to enumerate all patterns whose support is more than s,

$$S_{freq} = \{j \mid \sum_{i=1}^{n} I(T_j \subseteq G_i) \geq s\}. \tag{10}$$

On the other hand, what we need now is the *weighted substructure mining* to search for the best pattern in terms of the score

$$j^* = \arg\max_j \left| \sum_{i=1}^{n} w_i (2x_{ij} - 1) \right|, \tag{11}$$

where x_{ij} is defined as $x_{ij} := I(T_j \subseteq G_i)$, the weight for a training example is $w_i = u_i - \frac{1}{m}\sum_{k=1}^{m} u_k$, and the weight for a clearly negative example is $w_i = -v_i + \frac{1}{l}\sum_{k=1}^{l} v_k$.

The key idea of efficient graph mining is to exploit the *anti-monotonicity*, namely the frequency of a pattern is always smaller than or equal to that of its subgraph. In frequent substructure mining (10), one constructs a tree-shaped search space (i.e., DFS code tree) where each node corresponds to a pattern (Figure 2). The tree is generated from the root with an empty graph, and the pattern of a child node is made by adding one edge. As the pattern gets larger, the frequency decreases monotonically. If the frequency of the generated pattern T_j is s, it is guaranteed that the frequency of any supergraph of T_j is less than s. Therefore, the exploration is stopped there (i.e., *tree pruning*). By repeating node generation until all possibilities are checked, all frequent subgraphs are enumerated.

In the tree expansion process, it often happens that the generated pattern is isomorphic to one of the patterns that have already been generated. It leads to significant loss of efficiency because the same pattern is checked multiple times. The gspan algorithm solves this problem by the minimum DFS code approach, and we also adopted it for pruning isomorphic patterns.

In weighted substructure mining (11), the search tree is pruned by a different condition. Let us rewrite the weight as $w_i = y_i d_i$ where $d_i = |w_i|$ and $y_i = \mathrm{sign}(w_i)$. Then, the following bound is obtained: For any $T_j \subseteq T_k$,

$$\left| \sum_{i=1}^{n} w_i (2x_{ik} - 1) \right| \leq \gamma,$$

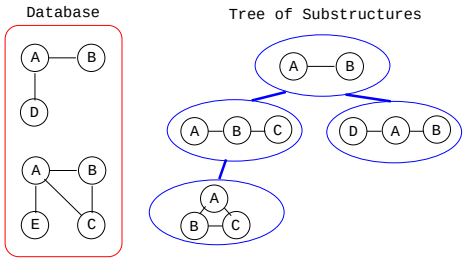


Fig. 2. Schematic figure of the tree-shaped search space of patterns (i.e., substructures)

where $\gamma = \max(\gamma^+, \gamma^-)$ and

$$\gamma^+ = 2 \sum_{\{i|y_i=+1, T_j \subseteq G_i\}} d_i - \sum_{i=1}^{n} d_i y_i,$$

$$\gamma^- = 2 \sum_{\{i|y_i=-1, T_j \subseteq G_i\}} d_i + \sum_{i=1}^{n} d_i y_i.$$

See [7] for the proof. When a pattern T_j is generated, the scores of its supergraphs T_k are upperbounded as above. Thus, if the upperbound is less than the current best value, we can safely quit further exploration.

4.1 Use of multiple constraints

In practice we found that the computation time of our algorithm is dominated by graph mining algorithm, so we propose to use multiple subgraphs for each iteration (multiple pricing[20]) in order to decrease the number of graph mining. This can be performed by mining top k subgraphs at each graph mining. In order to implement this change, we use a band of τ to maintain top k subgraphs which maximizes $|\sum_{i=1}^{m} u_i x_{ij} - \sum_{k=1}^{l} v_k \bar{x}_{kj}|$. It should be noted that no matter how many subgraphs we add for each iteration, the solution is kept optimal by solving linear programming [18].

5 QSAR Experiments

We used Endocrine Disruptors Knowledge Base (EDKB) data provided by the National Center for Toxicological Research[3] for measuring performance of our algorithms. Endocrine disruption is caused by the interference of the endocrine system by environmental or exogenous chemicals. The E-SCREEN assay of the EDKB consists of 59 molecules with activities provided in real number (logRPP).

[3] http://edkb.fda.gov/databasedoor.html

Table 1. 5-fold cross validation results on 59 molecules. For MGK, stopping probability 0.5 is chosen from $\{0.1 \dots 0.9\}$. $k = 3$ is chosen for kNN, and a ridge $1e-5$ is used for ridge regression. Proposed algorithm is stopped at 50 iteration.

Methods	l_1 error	l_2 error	time[s]	iterations	subgraphs
MGK + kNN	0.338±0.0326	0.240±0.0485	10.1	-	-
MGK + ridge	0.343±0.0282	0.213±0.0470	10.3	-	-
Proposed ($\epsilon = 0.01$)	0.291±0.0185	0.157±0.00701	30.7	10.3	9.6
Proposed ($\epsilon = 0.1$)	0.239±0.0136	0.107±0.00481	46.4	15.2	14.2
Proposed ($\epsilon = 0.2$)	**0.227±0.0124**	**0.101±0.00340**	139	37.6	14.4
Proposed ($\epsilon = 0.3$)	0.237±0.0124	0.123±0.00424	200	50	6.2
Proposed ($\epsilon = 0.5$)	0.282±0.0200	0.170±0.00887	178	50	2

We also used 61 clearly negative data, and set z to the lowest active level in the active data. The parameter C, which controls a generalization error, was set to 0.2. The performance was measured by 5 fold cross validation on Linux with Pentium4 2.4Ghz processor.

We compared our method with marginalized graph kernel (MGK) [1] in combination with ridge regression or kNN regression. MGK-based regression, however, cannot correctly include the information of clearly negative data, thus we just added them with labels as same value as the lowest active level in the active data. For comparison, we tried the same experimental setting on our graph regression algorithm, and denoted it as Proposed* in Table 2.

The results in the EDKB dataset are shown in Tables 1 and 2. For MGK with kNN regression or ridge regression, we can observe that inclusion of negative data degrades the performance. Performance of our our method was constantly better than MGK indifferent to regression algorithms suggests that our method better extracted the structurally characteristic patterns. Also good performances of "Proposed" over "Proposed*" in Table 2 validates our way of incorporating clearly negative data.

All the extracted subgraphs from 120 molecules are illustrated in Figure 3. Subgraphs are ordered from the top left to the bottom right according to their weights.

There are pros and cons both for classification and regression, i.e., classification involves a problem of discretization of activities while regression does not, but regression does not take into account clearly negative data. In this sense, our method is located between classification and regression.

While we built a regression model and found its component subgraphs, however, those extracted subgraphs are sometimes not so easy to interpret. For example, if a simple carbohydrate chain with fixed length is extracted, there are many ways of superimposing it on a molecule. Enriching atom and bond labels would be a better way to overcome this difficulty, and we are investigating this direction.

We can see from Figure 4 that the decrease in the number of iterations until convergence is almost in proportion to k, and we can see a similar curve for

Table 2. 5-fold cross validation results on 120 molecules. For MGK, stopping probability 0.5 is chosen from $\{0.1 \dots 0.9\}$. $k = 3$ is chosen for kNN, and a ridge $1e-5$ is used for ridge regression. Proposed algorithm is stopped at 50 iteration. Proposed* method regards 61 negative data labeled as lowest active value in the active data.

Methods	l_1 error	l_2 error	time[s]	iterations	subgraphs
MGK + kNN	0.474±0.0644	0.361±0.0469	29.0	-	-
MGK + ridge	0.454±0.0577	0.335±0.0431	29.1	-	-
Proposed ($\epsilon = 0.01$)	0.234±0.0114	0.100±0.00262	57.0	11.6	10.6
Proposed ($\epsilon = 0.1$)	**0.232±0.0108**	**0.087±0.00207**	112	21.6	20.6
Proposed ($\epsilon = 0.2$)	0.233±0.0132	0.101±0.00298	296	50	20.8
Proposed ($\epsilon = 0.3$)	0.249±0.0154	0.126±0.00420	289	50	15.6
Proposed ($\epsilon = 0.5$)	0.288±0.0201	0.163±0.00563	272	50	11.4
Proposed* ($\epsilon = 0.01$)	0.277±0.0191	0.153±0.00997	51.2	10	9
Proposed* ($\epsilon = 0.1$)	0.267±0.0182	0.128±0.00731	88.8	16.6	17.6
Proposed* ($\epsilon = 0.2$)	0.250±0.0139	0.104±0.00430	173	29.8	25.6
Proposed* ($\epsilon = 0.3$)	0.238±0.0124	0.106±0.00299	322	50	22.2
Proposed* ($\epsilon = 0.5$)	0.278±0.0205	0.147±0.00691	329	50	11.8

the time until convergence. The l_1 error and the number of subgraphs which contributed to the final ensemble almost did not change over different k, which guarantees the appropriate termination of the algorithm. The optimal number of k which let the learning algorithm converge the fastest depends on the data, but the LP theory gives validity of the final ensemble independent of the selection of k, and we observed no practical disadvantages just by setting k large.

6 Discussions

We have presented a graph regression algorithm using multiple subgraphs weighted by a linear programming. Experiments are carried out to show the usefulness of the algorithm in regression problems.

A method that first discovers all the subgraphs satisfying a certain condition, then classifies graphs by SVMs exists [21]. Our method, however, can discover subgraphs and add them to an ensemble simultaneously, therefore can save the cost of discovering subgraphs which satisfies some condition but do not contribute to the final ensemble. Also, knowledge based SVMs [22, 23] can take into account inequality constraints as well, but our algorithm is more efficient by the same reason above.

The importance of aligning functional groups in QSAR/QSPR is discussed in [4] and [5]. Alignment of pharmacophore was used to be done manually, but our algorithm is alignment-free, and might be useful for this problem.

Our algorithm automatically selects sparse features due to sparse regularizer. This has an advantage in interpretability, and we are not required to define or pre-register key structures [24], or to find a pre-image [25]. However, combining classical features such as partial charges, logP etc. might be useful to build a

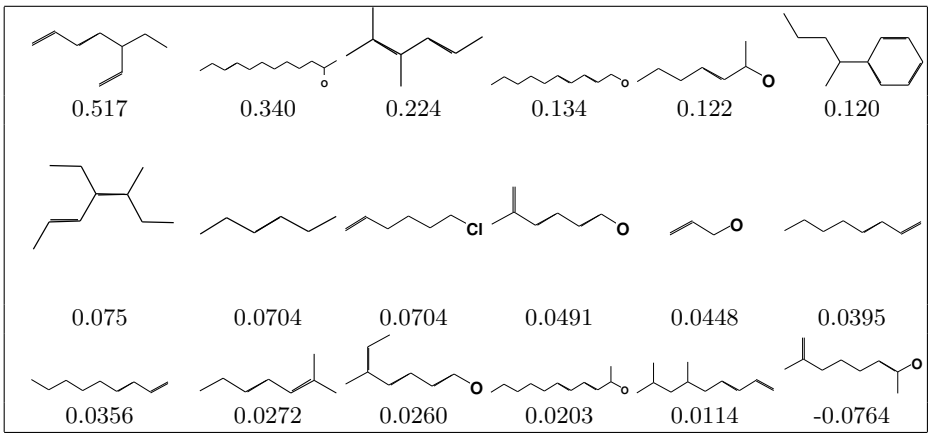


Fig. 3. Extracted subgraphs from 120 molecules. Subgraphs are ordered from the top left to the bottom right according to their weights. H atom is omitted, and C atom is represented as a dot for visual interpretability.

more precise model, and we are investigating this direction. Finally, we focused on mentioning properties and applications of our method in chemistry, but the framework of graph classification and regression is general, and can be applied to any data which consists of graphs.

Acknowledgments Computational resources were provided by the Bioinformatics Center, Institute for Chemical Research, Kyoto University, and the Supercomputer Laboratory, Kyoto University.

References

1. H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the twenty-first International Conference on Machine Learning*, pages 321–328. AAAI Press, 2003.
2. T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the sixteenth Annual Conference on Computational Learning Theory and seventh Kernel Workshop*, pages 129–143. Springer Verlag, 2003.
3. L. Ralaivola, S.J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Netw.*, 18(8):1093–1110, 2005.
4. H. Fröhrich, J. Wegner, F. Sieker, and Z. Zell. Kernel functions for attributed molecular graphs - a new similarity based approach to adme prediction in classification and regression. *QSAR & Combinatorial Science*, 25(4):317–326, 2006.
5. P. Mahé, L. Ralaivola, V. Stoven, and J.P. Vert. The pharmacophore kernel for virtual screening with support vector machines. Technical report, 2006. Technical Report HAL:ccsd-00020066.

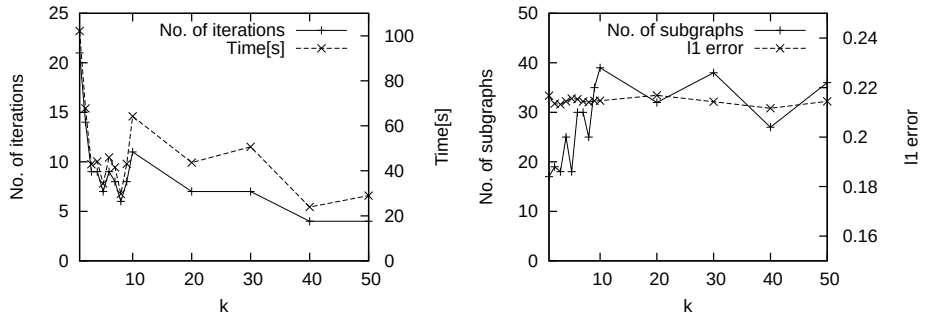


Fig. 4. Speed up of the algorithm by multiple pricing. We can see the number of boosting iterations and the time until convergence decreases in proportion to k (left), while keeping the number of subgraphs and l_1 error almost constant (right).

6. T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data minig*, pages 158–167. ACM Press, 2004.
7. T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In *Advances in Neural Information Processing Systems 17*, pages 729–736. MIT Press, 2005.
8. S. Kramer, L.D. Raedt, and C. Helma. Molecular feature mining in hiv data. In *Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, 2001.
9. L.D. Raedt and S. Kramer. The level-wise version space algorithm and its application to molecular fragment finding. In *Proceedings of the seventeenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 2001.
10. S. Kramer and L.D. Raedt. Feature construction with version spaces for biochemical applications. In *Proceedings of the eighteenth International Conference on Machine Learning*. AAAI Press, 2001.
11. A. Inokuchi. Mining generalized substructures from a set of labeled graphs. In *Proceedings of the fourth IEEE Internatinal Conference on Data Mining*, pages 415–418. IEEE Computer Society, 2005.
12. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 721–724. IEEE Computer Society, 2002.
13. T. Kadowaki, T. Adachi, T. Kudo, S. Okamoto, N. Tanaka, C.E. Wheelock, K. Tonomura, K. Tsujimoto, H. Mamitsuka, S. Goto, and M. Kanehisa. Chemical genomic study in endocrine disruptors on metabolic pathways. submitted, 2006.
14. R. Meir and G. Rätsch. An introduction to boosting and leveraging. In *Lecture Notes in Computer Science: Advanced lectures on machine learning*, pages 118–183, Heidelberg, 2003. Springer-Verlag.
15. A. Demiriz, K.P. Bennet, and J. Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1-3):225–254, 2002.
16. K. Tsuda and T. Kudo. Clustering graphs by weighted substructure mining. In *Proceedings of the twenty-third International Conference on Machine Learning*, pages 953–960. ACM Press, 2006.

17. J. Gasteiger and T. Engel. *Chemoinformatics: a textbook*. Wiley-VCH, 2003.
18. G. Rätsch, A. Demiriz, and K.P. Bennett. Sparse regression ensembles in infinite and finite hypothesis spaces. *Machine Learning*, 48(1-3):189–218, 2002.
19. B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.
20. D. G. Luenberger. *Optimization by Vector Space Methods*. Wiley, 1969.
21. S. Kramer, E. Frank, and C. Helm. Fragment generation and support vector machines for including sars. *SAR and QSAR in Environmental Research*, 13(5):509–523, 2002.
22. O.L. Mangasarian and E.W. Wild. Knowledge-based kernel approximation. *Journal of Machine Learning Research*, 5:1127–1141, 2004.
23. Q.V. Le, A.J. Smola, and T. Gärtner. Simpler knowledge-based support vector machines. In *Proceedings of the twenty-third International Conference on Machine Learning*. ACM Press, 2006.
24. C.A. James, D. Weininger, and J. Delany. Daylight theory manual, 2004.
25. T. Akutsu and D. Fukagawa. Inferring a graph from path frequency. In *Proceedings of the sixteenth Annual Symposium on Combinatorial Pattern Matching*, pages 371–382. Springer, 2005.

Matching Based Kernels for Labeled Graphs

Adam Woźnica and Alexandros Kalousis and Melanie Hilario

University of Geneva, Computer Science Department
Rue General Dufour 24, 1211 Geneva 4, Switzerland
{woznica, kalousis, hilario}@cui.unige.ch

Abstract. For various classification problems it is most natural to represent training examples as labeled graphs. As a result several kernel functions over these complex structures have been proposed in the literature so far. Most of them exploit the Cross Product Kernel between two sets resulting from the decompositions of corresponding graphs into subgraphs of a specific type. The similarities between the substructures are often computed using the $0-1$ Kronecker Delta Kernel. This approach has two main limitations: (i) in general most of the subgraphs will be poorly correlated with the actual target variable, adversely affecting the generalization of a classifier and (ii) as no graded similarities on subparts are computed, the expressivity of the resulting kernels is reduced. To tackle the above problems we propose here a class of graph kernels based on set distance measures whose computation is based on specific pairs of points from the corresponding graph decompositions. The actual matching of elements from the two sets depends on a graded similarity (standard Euclidean metric in our case) between the elements of the two sets. To make our similarity measure positive semidefinite we exploit the notion of the proximity space induced by a given set distance measure. To practically demonstrate the effectiveness of our approach we report promising experimental results for the task of activity prediction of drug molecules.

1 Introduction

Support Vector Machines (SVMs), and Kernel Methods in general, are becoming increasingly popular for their performance [1]. As most of the real-world data can not be easily represented in an attribute-value format many kernels for various kinds of structured data have been defined in the literature. In particular graphs are a widely used tool for modeling structured data in the machine learning community and many kernels over these complex structures have been proposed so far. These kernels have been mainly applied for predicting the activity of chemical molecules represented by undirected labeled graphs where vertices are atoms and edges are covalent bonds.

However, due to the powerful expressiveness of graphs it has been proved that kernels over arbitrarily structured graphs, taking their full structure into account, can be neither computed [2] nor even approximated efficiently [3]. The most popular approach to tackle the above problem is based on a decomposition of graphs into particular subparts which are compared via subkernels. The substructures considered are mainly walks [2, 4–7], however, other researchers have experimented with shortest paths [8],

subtrees [3], cyclic and tree patterns [9] and limited-size general subgraphs centered at each vertex [10].

The existing kernels based on decompositions have two main limitations. First, most of them combine substructures using the Cross Product Kernel which takes *all* the possible substructures of a given type into account. This might adversely affect the generalization of a classifier since most of the subgraphs, and hence attributes in the feature space, will be poorly correlated with the actual class variable. Second, almost all the existing graph kernels use the $1-0$ matching kernel to compute similarities between subgraphs. This means that the expressivity of these kernels is reduced since they are not able to find partial similarities.

To tackle the above problems we propose here a class of graph kernels that: (i) are based on specific matchings of substructures (walks in our case) [1] from the corresponding decompositions of two graphs and (ii) compute graded similarities between these subparts. More precisely, we exploit kernels in the proximity space induced by set distance measures where the mapping is defined by a given representation set [11]. The set distance measures we experimented with focus only on specific pairs of points from the two sets of decompositions. The actual matching of the elements of the corresponding sets depends on the pairwise graded similarities with are computed by means of the standard Euclidean metric. Finally, all the considered distances are computable in a polynomial time which means that these kernels can be applied to large graph databases. To practically demonstrate the effectiveness of our approach we report experimental results for the task of activity prediction of drug molecules for the Mutagenesis and the Carcinogenicity datasets.

2 Primer of graph theory

An undirected *graph* $G = (\mathcal{V}, \mathcal{E})$ is described by a finite set of *vertices* $\mathcal{V} = \{v_1, \ldots, v_n\}$ and a finite set of *edges* $\mathcal{E} = \{e_1, \ldots, e_m\}$ such that $\mathcal{E} = \{\{v_i, v_j\} : v_i, v_j \in \mathcal{V}\}$. For *labeled* graphs there is additionally a set of labels $\mathcal{L}$ together with a function $label : \mathcal{V} \cup \mathcal{E} \rightarrow \mathcal{L}$ that assigns a label to each vertex and edge. We denote a graph database as $\mathcal{G}$. The *adjacency* matrix A of G is defined as $[A]_{ij} = 1 \iff \{v_i, v_j\} \in \mathcal{E}$ and $[A]_{ij} = 0$ otherwise.

Some special graphs relevant to the existing graph kernels are walks, paths and trees. A *walk* in a graph G is a sequence of vertices $w = [v_1, v_2, \ldots, v_{s+1}]$ such that $v_i \in \mathcal{V}$ for $1 \leq i \leq s+1$ and $\{v_i, v_{i+1}\} \in \mathcal{E}$ for $1 \leq i \leq s$. We define a length $l(w)$ of a walk as the number of vertices and edges in this sequence, i.e. in the above example $l(w) = 2s + 1$. A *path* is a walk that never goes through the same edge twice. A *tree* is a connected, acyclic graph.

3 Motivation and Related work

As mentioned in Section 1 the idea of decomposition of graphs into subparts is reflected in most kernels for these type of structures [2, 4, 5, 8–10, 3, 6]. This mechanism origi-

[1] More precisely we focus here on walks without consecutive repetitions of the same cycles of nodes. It should be stressed, however, that any type of subgraphs can be used.

nates from kernels for general structured data where the computation of the similarity between two complex objects is based on the similarities of objects' parts computed by means of subkernels [12, 13].

An integral part of the above kernels for graphs is the *decomposition* of these structures into a *multi-set* of its parts (walks, paths, trees, cyclic pattern, etc.) and the final kernel is defined as the *Cross Product Kernel* between the corresponding multi-sets of decompositions. For particular decompositions $\{g\} = \mathcal{G}_t$ and $\{g'\} = \mathcal{G}'_t$ (t denotes that these subgraphs are of type t) of graphs G and G', respectively, the above kernel can be written as:

$$K(G, G') = \sum_{g \in \mathcal{G}_t, g' \in \mathcal{G}'_t} k(g, g') \tag{1}$$

where k is a kernel over specific types of graphs $\mathcal{G}_t$ and the summation over the elements of the multisets takes into account their multiplicity.

From Equation 1 it is clear that *all* the possible subgraphs of a given type are matched by means of a subkernel. This might adversely affect the generalization of a large margin classifier since due to the combinatorial growth of the number of distinct subgraphs most of the features in the feature space will be poorly correlated with the target variable [14, 10]. Possible solutions to this problem include down-weighting of the contribution of larger subgraphs [15, 2], using prior knowledge to guide the selection of relevant parts [16] or considering contextual information for limited-size subgraphs [10]. The other solution would be to change the right side of Equation 1 such that the sum runs over specific elements of the corresponding sets excluding elements which are likely to be irrelevant for the given target variable. This kernel will be directly based on specific pairs of elements from the two sets and can be written in a general form as:

$$K \ : \ K(G, G') = f(\{k(g, g') | (g, g') \in \mathcal{G}_t \times \mathcal{G}'_t\}) \tag{2}$$

i.e. it is some function of the set of pairwise elementary kernels, $k(g, g')$ where $(g, g') \in \mathcal{G}_t \times \mathcal{G}'_t$. The idea of using specific pairs of points in a set kernel is promising, however, it is easy to see that the kernel from Equation 2 is not positive semidefinite (PSD) in general.

The other problem with general kernels based on decompositions is that most of them can only use the Kronecker Delta Kernel (i.e. $k_\delta(x, y) = 1 \iff x = y, k_\delta(x, y) = 0$ otherwise) on subgraphs. As a result the ability to find partial similarities is lost and the expressivity of these kernels is reduced. A graded similarity on walks is considered in the kernel from [6], however, it suffers from high computational complexity since it requires taking powers of the adjacency matrix of the direct product graph, leading to huge runtime and memory requirements [8]. Finally, some of the kernels based on walks suffer from the problem known as *tottering*, i.e. by iteratively visiting the same cycle of nodes, small identical substructures in input graphs can lead to high similarity scores. This problem was recognized in [5] where the authors proposed a modification of the algorithm from [4] modifying the underlying random walk model, however, their algorithm did not lead to a significant improvement in performance.

To overcome the above problems we propose a class of set kernels that are based on set distance measures. The final dissimilarity in the considered set distances is based directly on specific pairs of points from two sets. Encouraged by the effectiveness of

walks in chemical domains we focus here on these type of structures (without repetitions of the same cycles of nodes) constructed from a depth first exploration emanating from each node in a graph and yielding all the walks of length l [7]. All the considered distances are computable in a polynomial time which means that our kernel can be applied to large graph databases. The actual matching of the elements of the corresponding sets depends on the pairwise graded similarities which are computed by the standard Euclidean metric. To make the final kernel PSD we define it in the proximity space induced by set distance measures where the mapping is defined by a given representation set [11]. Mapping to a proximity spaceis a classical way to make a PSD kernel from measures of similarity or distances.

It should be noted that in parallel to kernels based comparison of particular subparts there exist other approaches for controlling computational complexity of graph kernels. One line of research focuses on special kinds of graphs such as strings, trees and nodes in graphs [17]. The resulting kernels are efficient, however, they loose most of the modeling power of general graphs. An alternative direction explicitly controls the dimensionality of the feature space by generating sets of connected graphs that occur frequently as subgraphs in the graph database and this frequency is beyond a user defined threshold [18, 19]. These kernels form an attractive alternative to kernels based on decompositions since the corresponding feature space is constructed explicitly. On the other hand these methods bear difficulties since their efficiency is threshold-dependent.

4 Kernels on Graphs

In this section we define a class of kernels based on matchings for labeled graphs. The construction of these kernels is based on: (i) set distance measures which will be described in Section 4.1 and (ii) kernels in proximity spaces induced by set distance measures (presented in Section 4.2).

4.1 Distances on Sets

A number of different measures have been proposed in the literature for defining distances between sets of objects. We will briefly present some of them. Consider two sets $A = \{a_i\} \subseteq \mathcal{X}$ and $B = \{b_j\} \subseteq \mathcal{X}$. Let $d(.,.)$ be a metric defined on $\mathcal{X}$. The set distance measure D defined on $2^{\mathcal{X}}$ as: $D \ : \ D(A,B) = f(\{d(a_i,b_j)|(a_i,b_j) \in A \times B\})$, i.e. is some function of the pairwise distances, $d(a_i,b_j)$, of the set of all pairs $(a_i,b_j) \in A \times B$. A and B should be nonempty and finite sets. Within this framework we can define the following set distance measures. The *(Normalized) Average Linkage*,$\mathrm{D_{AL}}$, is defined as the (normalized) average of all pairwise distances, $D_{AL}(A,B) = \frac{\sum_{ij} d(a_i,b_j)}{|A||B|}$. The *Sum of Minimum Distances*, $\mathrm{D_{SMD}}$, discussed in [20], is the sum of the minimum distances of the elements of the first set to the elements of the second set and vice versa, normalized be the sum of cardinalities of the two sets, $D_{SMD}(A,B) = \frac{1}{|A|+|B|}(\sum_{a_i} \min_{b_j}\{d(a_i,b_j)\} + \sum_{b_i} \min_{a_j}\{d(b_i,a_j)\})$. The *Hausdorff* distance measure, $\mathrm{D_H}$, discussed in [20], is one of the best known distances measures between sets. By definition, two sets A and B are within the *Hausdorff* distance D of each other iff every point of A is within distance D of at least one point

of B and vice versa. The ***RIBL*** distance, $\mathrm{D_{RIBL}}$, is the sum of the minimum distances of the elements of the smaller set to the elements of the larger, [21]. Formally if $|A| < |B|$ it is defined as $D_{RIBL}(A,B) = \frac{\sum_{a_i} \min_{b_j}\{d(a_i,b_j)\}}{|B|}$ and if $|A| \geq |B|$ then $D_{RIBL}(A,B) = \frac{\sum_{b_j} \min_{a_i}\{d(a_i,b_j)\}}{|A|}$.

Another family of more elaborate distance measures is based on the definition of a set of relations $R = \{R_i | R_i \subseteq A \times B\}$ between the two sets. The computation of the distance measure will be based on an $R_i \in R$ that minimizes the sum of distances computed on the elements that are part of the relation R_i. In the *Surjections*, $\mathrm{D_S}$, set distance measure the set of relations R consists of all the possible surjections of the larger to the smaller set [20]. In the *Linkings*, $\mathrm{D_L}$, distance measure the set of relations is the set of all possible linkings [20]. A linking is a mapping of one set to the other where all elements of each set participate in at least one pair of the mapping. For *Fair Surjections*, $\mathrm{D_{FS}}$, distance measure the set of relations is the set of all fair surjections, [20]. A surjection is fair if it maps as evenly as possible the elements of the larger set to the elements of the smaller set. For the above set distance measures the final distance of the two sets is defined as the minimum sum, over all R_i, of the distances of the pairs of elements that participate in the surjection, linkings and fair surjections, respectively: $D_{S \vee L \vee FS}(A,B) = \frac{\min_{R_i \in R} \sum_{(a_i,b_j)\in R_i} d(a_i,b_j)}{|R_i|}$. In the *Matchings*, $\mathrm{D_M}$, the set of all possible matchings is considered within which the minimum distance is computed [22]. In a matching each element of the two sets is associated with *at most* one element of the other set. This distance is given by: $D_M(A,B) = \min_{R_i \in R}(\sum_{(a_i,b_j)\in R_i} d(a_i,b_j) + (|B - R_i(A)| + |A - R_i^{-1}(B)|) \times \frac{M}{2})$ where M is the maximum possible distance between two elements. $\mathrm{D_M}$ is normalized as $D_M(A,B) := \frac{2D_M(A,B)}{D_M(A,B)+(|A|+|B|)/2}$.

It should be noted that all of the set distance measures defined above take values between 0 and 1. In the rest of the paper we will focus only on the $\mathrm{D_{SMD}}$, $\mathrm{D_H}$, $\mathrm{D_{RIBL}}$, $\mathrm{D_S}$, $\mathrm{D_L}$, $\mathrm{D_{FS}}$ and $\mathrm{D_M}$ set distance measures since these are the ones based only on specific pairs of elements. The $\mathrm{D_{AL}}$ will be examined only in comparative studies.

4.2 Set Kernels in the Proximity Space

To define a PSD kernel using the set distance measures from Section 4.1 we will use the proximity space representation [11]. This space is defined by a given set distance measure and a representation set (set of prototypes) of learning instances. More precisely, given a representation set $S = \{s_1, \ldots, s_n\} \subseteq 2^{\mathcal{X}}$ and a set distance measure $D : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \to \mathbb{R}_0^+$ we define a mapping $D(z,S) : 2^{\mathcal{X}} \to \mathbb{R}^n$ as $D(z,S) = [D(z,s_1), \ldots, D(z,s_n)]^T$ where $z \in 2^{\mathcal{X}}$ (here we assumed that our learning examples are sets of objects). Since distance measures are nonnegative, all the data examples are projected as points to a nonnegative orthotope of that vector space. The dimensionality of this space is controlled by the size of the set S (usually the full training set).

The construction of the proximity space is justified by the fact that for an object s_i belonging to the same class as z, $D(z,s_i)$ should be small while for an object s_j of different classes $D(z,s_j)$ should be large, resulting in a set of features with possibly high discrimination power [11]. On the other hand if s_i is a characteristic object of a

particular class, then the feature $D(z, s_i)$ has a large discrimination power while for s_j being an outlier $D(z, s_j)$ may discriminate poorly. The definition of a set kernel in the proximity space amounts to choosing a set distance measure, D, and a vectorial kernel, k, in the induced space. The resulting Gram matrix of the set kernel $\mathrm{K_P}$ consists of the elements: $(\mathbf{K}_{P_D})_{ij} = k(D(z_i, S), D(z_j, S))$.

4.3 Matching Based Kernels

In this section we combine set distance measures and kernels defined in proximity spaces to define matching based kernels for graphs. To exploit set distance measures we assign to each graph G the set of subgraphs obtained from a particular decomposition of G into subparts. In this work a graph will be decomposed into a set of walks (without repetitions of the same two-nodes-cycles) obtained from a depth first exploration emanating from each node in a graph and yielding all the walks of length l [7] [2]. In particular for $l = 1$ a molecule is represented as a set of atoms; for $l = 2$ a compound is decomposed into a set of pairs with the first element being an atom and the second element being one of its adjacent bonds. It should be noted that for a molecule with n atoms and m bonds, the complexity of extracting all the walks of length up to l is at most $O(n\alpha^l)$ where α is the branching factor (slightly above two in organic chemistry) [7].

Given two decompositions into walks $\{g\} = \mathcal{G}$ and $\{g'\} = \mathcal{G}'$ of graphs G and G' the matching based kernel on graphs can be written as:

$$K_{P_D}(G, G') = k(D(\mathcal{G}, S), D(\mathcal{G}', S)) \tag{3}$$

where D, k, S denote the set distance measure, the elementary kernel and the representation set, respectively.

5 Experiments

We will experiment on two graph classification problems: Mutagenesis and Carcinogenicity. The Mutagenesis dataset was introduced in [23]. The application task is the prediction of mutagenicity of a set of 188 aromatic and heteroaromatic nitro-compounds which constitute the "regression friendly" version of this dataset. The other classification problem comes from the Predictive Toxicology Challenge and is defined over carcinogenicity properties of chemical compounds [24]. This dataset lists the bioassays of 417 chemical compounds for four type of rodents: male rats (MR), male mice (MR), female rats (FR) and female mice (FM) which give rise to four independent classification problems. We transformed the original dataset (with eight classes) into a binary problem by ignoring EE (equivocal evidence), E (equivocal) and IS (inadequate study) classes, grouping SE (some evidence), CE (clear evidence) and P (positive) in the positive class and N (negative) and NE (no evidence) in the negative one.

[2] As presented in Section 2 the length of a walk is defined as the number of its constituent vertices and edges.

In the experiments we want to perform several comparisons of the SVM and kNN algorithms with different distances over sets of subgraphs and using different decompositions based on walks. First, for various set distance measures we want to explore the performance of the linear kernel defined in the corresponding proximity spaces for various lengths of walks. This kernel will be used with the SVM method, resulting in the $\mathrm{SVM_P}$ algorithm. The goal of this experiment is to examine the influence of walk lengths to the performance of the classifier. Second, we want to check how the performance of SVM with the above kernels compares with SVM with the following two set kernels taking all the elements of the two decompositions into account (i) the Cross Product Kernel (CPK) with the linear kernel as an elementary kernel and (ii) the linear kernel in the proximity space induced by the $\mathrm{D_{AL}}$ distance measure; kernels matching all subgraphs are a standard way of tackling classification problems where instances are represented as graphs. Third, we are going to examine how the $\mathrm{SVM_P}$ algorithm compares with the kNN algorithm where these distances are used directly. By doing this we establish whether $\mathrm{SVM_P}$ indeed provides an improvement over the simple kNN algorithm. Finally, we would like to examine performance of the SVM algorithm where a given molecule will be represented by a set of *all* walks of length smaller than (or equal to) a given value. The reason we use the linear kernel in the experimental setup is to make a fair comparison between the algorithms and to avoid the situation where an implicit mapping given by a nonlinear kernel will influence the results.

For $\mathrm{SVM_P}$ and the SVM with the CPK the regularization parameter C was optimized in an inner 10-fold cross validation loop over the set $C = \{0.1, 1, 10, 50\}$ whereas for kNN algorithm the number of nearest neighbors was optimized over the set $\{1, 3, 9\}$. In all the experiments accuracy was estimated using stratified ten-fold cross-validation and controlled for the statistical significance of observed differences using McNemar's test (sig. level=0.05).

6 Results

The first dimension of comparison is to examine the influence of lengths of walks to the predictive performance of the SVM algorithm. The results for walks of length up to eleven are presented in Figure 1 [3]. From these results it is clear that the optimal length of walks depends on the actual application and the set distance measure. However, the following findings can be observed. First, for all the datasets and for all the set distance measures (with the exception of FairSurjections and Hausdorff distance measures) the highest predictive accuracy is obtained for walks of lengths between five and seven. Second, in Carcinogenicity for walks longer than seven a decay in performance is observed. Additionally, the poor performance of the FairSurjections can be noted which might be explained by the fact that FairSurjections distance measure maps as evenly as possible the elements of the larger set to the elements of the smaller set which might result in a situation where irrelevant subgraphs from the two decompositions will be matched, affecting the performance of a classifier.

[3] Due to the space limitation the results are only given for the Mutagenesis and the MR version of the Carcinogenicity datasets. For other versions of the latter dataset similar trends hold.

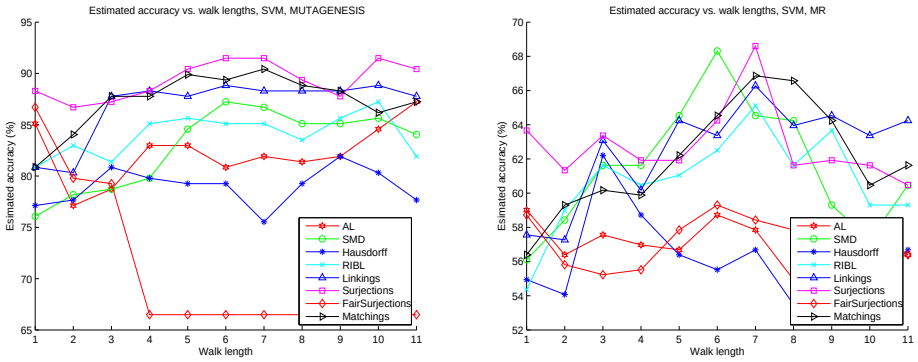


Fig. 1. Estimated accuracy vs. lengths of walks for different set distance measures in the Mutagenesis and Carcinogenicity (MR) datasets.

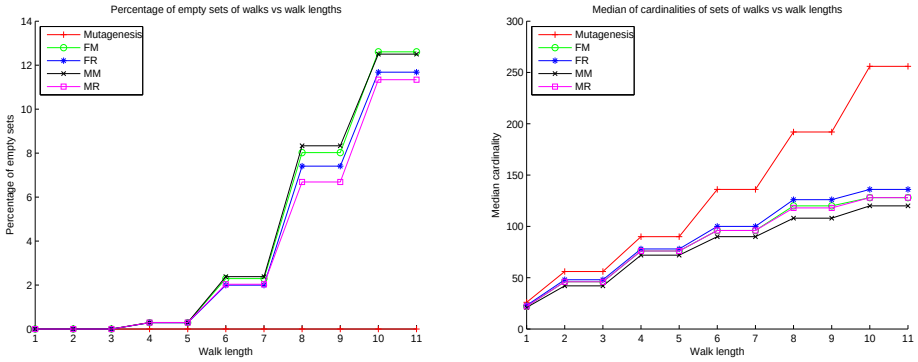


Fig. 2. Percentage of empty sets and medians of cardinalities vs. lengths of walks for the Mutagenesis and Carcinogenicity datasets.

One problem with the decomposition of molecules into set of walks where consecutive repetitions of the same cycles of nodes are not allowed is that small molecules for high values of l will be associated with an empty set of walks. To get more insight into this problem the first graph in Figure 2 presents the dependency between the lengths of the considered walks and the percentage of molecules associated with empty sets. In particular in the Carcinogenicity datasets for walks of length equal to seven only six molecules are associated with empty sets. As a result one can expect that the performance of a classifier operating on such incomplete descriptors will be harmed. Indeed, a correlation between the percentage of empty set descriptors and the performance of SVM for walks longer than seven can be observed. Based on the above discussion and on the results from Figure 1, we will focus in our consecutive experiments on walks of length seven [4].

[4] As mentioned the optimal walk length will probably depend on the actual application and ideally should be learned as well. It means that the reported performance might be biased compared to the true generalization error, as we did not use training/validation partitions for the walk length optimization.

Table 1. Accuracy and significance test results of $\mathrm{SVM_P}$ and kNN for the Mutagenesis and Carcinogenicity datasets where walks of length seven are considered (+ stands for a significant win of the first algorithm in the pair, - for a significant loss and = for no significant difference). The first sign in the parenthesis corresponds to the comparison of $\mathrm{SVM_P}$ vs. kNN, the second to $\mathrm{SVM_P}$ vs. SVM with Cross Product Kernel and the last one to $\mathrm{SVM_P}$ vs. $\mathrm{SVM_P}$ with $\mathrm{K_{P_{AL}}}$. Due to the space limitation for the MM dataset only the results for $\mathrm{SVM_P}$ are reported. Additionally the default accuracy (*Def.*) for each dataset is given.

	Mutagenesis		*FR*		*FM*		*MR*		*MM*
Dist.	$\mathrm{SVM_P}$	kNN	$\mathrm{SVM_P}$	kNN	$\mathrm{SVM_P}$	kNN	$\mathrm{SVM_P}$	kNN	$\mathrm{SVM_P}$
$\mathrm{D_{SMD}}$	86.7 (===)	81.4	64.7 (===)	64.1	62.7 (+==)	55.6	64.5 (==+)	59.6	67.0 (+==)
$\mathrm{D_H}$	75.5 (=-=)	79.8	65.0 (=- -)	65.0	64.2 (===)	65.0	56.7 (===)	54.4	62.8 (===)
$\mathrm{D_{RIBL}}$	85.1 (+==)	70.2	64.7 (=- -)	64.4	60.5 (+-=)	52.7	65.1 (+=+)	51.7	62.2 (===)
$\mathrm{D_S}$	91.5 (+++)	83.0	63.5 (===)	64.1	61.0 (+==)	52.1	68.6 (+++)	54.1	63.4 (===)
$\mathrm{D_L}$	88.3 (+=+)	82.4	65.5 (===)	64.1	59.3 (=-=)	54.4	66.3 (+++)	58.7	65.2 (+==)
$\mathrm{D_{FS}}$	66.5 (=- -)	66.5	67.5 (===)	64.1	62.7 (===)	58.7	58.4 (===)	58.7	62.5 (===)
$\mathrm{D_M}$	90.4 (+++)	72.9	67.2 (===)	64.4	61.0 (===)	56.4	66.9 (+++)	57.0	62.5 (+==)
CPK	*83.0*		*67.5*		*63.6*		*58.4*		*63.4*
$\mathrm{K_{P_{AL}}}$	*81.9*		*67.2*		*63.0*		*57.8*		*65.5*
Def.	*66.5*		*65.5*		*59.0*		*55.8*		*61.6*

The next dimension of comparison is the relative performance between $\mathrm{SVM_P}$ and SVM with kernels based on all the elements of the two decompositions (the latter, as already mentioned, is a standard approach to tackle graph problems). We experimented with the following kernels in this category: the Cross Product Kernel (CPK) and the linear kernel in the proximity space induced by the $\mathrm{D_{AL}}$ distance measure ($\mathrm{K_{P_{AL}}}$). The main point of this comparison is to examine whether there are cases in which different ways of matching the elements of two sets of subgraphs can be more beneficial than the standard averaging which matches everything with everything. The results (with the significance test results in parenthesis) are presented in Table 1. From the results it is clear that the relative performance of kernels based on specific pairs of elements and kernels based on averaging depends on the actual application. The strongest advantage of the former is in the Mutagenesis and the MR datasets whereas the opposite trend holds for the remaining datasets. Overall the choice of the appropriate way of matching the elements of two sets depends on the application and ideally should be guided by domain knowledge, if such exists. Nevertheless, the relative performance of the different kernels provides valuable information about the type of problem we are facing. For example examining Mutagenesis and Carcinogenicity we see that although they correspond to the same type of classification problem, i.e. classification of graphs, in the latter (except for the MR) averaging works better, hinting that the global structure of the molecules is important, whereas in the former averaging performs poorly, indicating that matching specific components of the molecules is more informative. It should be also noted that in all the datasets the differences between CPK and $\mathrm{K_{P_{AL}}}$ were not statistically meaningful.

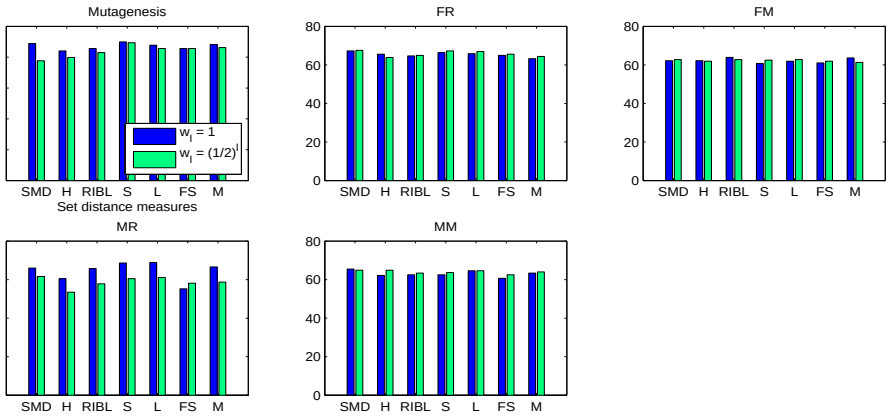


Fig. 3. Accuracy of the $\mathrm{SVM_P}$ algorithm for the Mutagenesis and Carcinogenicity datasets where all walks of lengths smaller than (or equal to) eleven are used. Two different weighting schemas are considered: $w_l = 1$ denotes that walks of different length have equal importance whereas $w_l = (\frac{1}{2})^l$ characterizes the algorithm where longer walks are exponentially down-weighted.

We also compared the performance of a standard kNN algorithm on the standard set distance measures with that of $\mathrm{SVM_P}$ in order to establish whether the latter indeed brings some improvement over the naive way of exploiting distances (these results are also listed in Table 1). Indeed kNN was never significantly better than $\mathrm{SVM_P}$ and it was significantly worse in fourteen out of thirty five cases. The better performance of $\mathrm{SVM_P}$ in the proximity space in comparison with the "standard" kNN can be explained by the fact that working in the proximity space gives a more global view to the data. More precisely the kNN algorithm in a non proximity space examines a single neighborhood of a given instance which is to be classified. On the other hand $\mathrm{SVM_P}$ and $\mathrm{kNN_P}$ in the proximity space have access to all the neighborhoods of points in the representation set. These neighborhoods are precisely given by the instances in the new space.

Finally, we examined the performance of the $\mathrm{SVM_P}$ classifier where a given molecule is represented by a set of *all* walks of length smaller than a given value (eleven in our case). More precisely a compound C_1 is represented by eleven sets $[S_1, \ldots, S_{11}]$ where the set $S_l, 1 < l < 11$ contains all possible walks of length l. The distance between compound C_1 and C_2 (we assume that C_2 has the set representation of $[T_1, \ldots, T_{11}]$) is given as $\mathcal{D}(C_1, C_2) = \sum_{l=1}^{11} w_l D(S_l, T_l)$, where D is a set distance measure and w_l are weights. Here we experimented with two different weighting schemas: (i) $w_l = 1, 1 \leq l \leq 11$, i.e. walks of different length have equal importance to the final distance and (ii) $w_l = (\frac{1}{2})^l, 1 \leq l \leq 11$, i.e. longer walks are exponentially down-weighted. The results are presented in Figure 3. The main observation is that for some datasets (Mutagenesis and MR) the down-weighting of longer walks harms the performance of $\mathrm{SVM_P}$; in Mutagenesis the down-weighting schema was never significantly better and it was significantly worse in two cases whereas in MR it was significantly worse in five cases (the significance tests are not presented in Figure 3). For other datasets the difference in accuracy was not statistically significant. The above finding is in contrast to the common practice in various graph kernels [15, 2] and it might indicate that for some

applications also larger subgraphs are correlated with the target variable. The other observation is that, the performance of the representation based on all walks up to a given length is comparable to the one where the lengths were fixed to a specified value.

To situate the performance of our relational learner to other relational learning systems we give the best results reported in the literature on the same benchmark datasets. For the Mutagenesis dataset we obtained 91.5 % accuracy (for the Surjections distance) while the best result from the literature (estimated with ten fold cross-validation) was 90.4 % [5]. The results for the Carcinogenicity dataset are not directly comparable with other results from the literature since different evaluation metric was used (accuracy instead area under ROC curve).

7 Conclusions and Future Work

It has been argued in [6] that a "good" kernel for graphs should fulfill at least the following requirements: (i) should be a good similarity measure for graphs, (ii) its computational time should be possible in polynomial time, (iii) should be positive semidefinite and (iv) should be applicable for various graphs. In this paper we propose a class of kernels for graphs which are based on walks without repetitive occurrences of nodes, that are computable in polynomial time, that are positive semidefinite and are applicable to a wide range of graphs. Additionally the distinctive feature of our graph kernel is that, instead of taking into account all the possible subgraphs of a given type, it matches only specific subparts. More precisely we proposed a class of kernels for graphs which directly exploit the set distance measures. These kernels are defined in the proximity space induced by set distance measures where the mapping is defined by a given representation set. To practically demonstrate the effectiveness of our approach we report experimental results for the task of activity prediction of drug molecules.

There are several possible directions for further work. First, we would like to experiment with non-PSD kernels such as the one from Equation 2. This is possible since recent experimental and theoretical results state that even a kernel which is non-PSD can be plugged into the SVMs. Second, we will experiment with decompositions of graphs into substructures other than walks (e.g. the cyclic patterns from [9]). For such subgraphs the standard Euclidean metric can not be used directly and other, more general distances on graphs (e.g. the graph edit distance) need to be considered. Third, we are going to compare our methods with alternative ways of creating smoother feature space as mentioned in Section 3. Finally we would like to avoid empty sets in the set of subgraphs representation of molecules as shown in Section 6. In particular we are going to experiment with mining frequent (connected) subgraphs approach as presented in [18, 19].

References

1. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press (2004)
2. Gärtner, T., Flach, P., Wrobel, S.: On graph kernels: Hardness results and efficient alternatives. In: Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop. (2003)

3. Ramon, J., Gärtner, T.: Expressivity versus efficiency of graph kernels. In: First International Workshop on Mining Graphs, Trees and Sequences (help with ECML/PKDD'03). (2003)
4. Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In: Proceedings of 20th International Conference on Machine Learning (ICML-2003), Washington, DC (2003)
5. Mahé, P., Ueda, N., Akutsu, T., Perret, J.L., Vert, J.P.: Extensions of marginalized graph kernels. In: ICML 2004. (2004)
6. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics **21**(1) (2005) i47–i56
7. Ralaivola, L., Swamidass, S.J., Saigo, H., Baldi, P.: Graph kernels for chemical informatics. Neural Networks (2005) 1093–1110
8. Borgwardt, K.M., Kriegel, H.P.: Shortest-path kernels on graphs. In: Proceedings of the 5th IEE Internatinal Conference on Data Mining. (2005)
9. Horváth, T., Gärtner, T., Wrobel, S.: Cyclic pattern kernels for predictive graph mining. In: Proceedings of KDD'04. (2004)
10. Menchetti, S., Costa, F., Frasconi, P.: Weighted decomposition kernels. In: Proceedings of 22nd International Conference on Machine Learning (ICML-2005). (2005)
11. Pekalska, E., Paclík, P., Duin, R.P.: A generalized kernel approach to dissimilarity-based classification. Journal of Machine Learning Research **2** (2001) 175–211
12. Haussler, D.: Convolution kernels on discrete structures. Technical report, UC Santa Cruz (1999)
13. Woźnica, A., Kalousis, A., Hilario, M.: Kernels over relational algebra structures. In: The Ninth Pacific-Asia Conference on Knowledge Discovery and Data, Hanoi, Vietnam (2005)
14. Ben-David, S., Eiron, N., Simon, H.: Limitations of learning via embeddings in euclidean half spaces. Journal of Machine Learning Research **3** (2002) 441–461
15. Collins, M., Duffy, N.: Convolution kernels for natural language. In Dietterich, T.G., Becker, S., Ghahramani, Z., eds.: Advances in Neural Information Processing Systems 14, Cambridge, MA, MIT Press (2002)
16. Cumby, C., Roth, D.: On kernel methods for relational learning. In: Proceedings of 20th International Conference on Machine Learning (ICML-2003), Washington, DC (2003)
17. Gärtner, T.: A survey of kernels for structured data. SIGKDD Explor. Newsl. **5**(1) (2003) 49–58
18. Deshpande, M., Kuramochi, M., Karypis, H.: Frequent sub-structure-based approaches for classifying chemical compounds. In: Proceedings of ICDM'03. (2004)
19. Kramer, S., Readt, L., Helma, C.: Molecular feature mining in hiv data. In: Proceedings of KDD'01. (2001)
20. Eiter, T., Mannila, H.: Distance measures for point sets and their computation. Acta Informatica **34**(2) (1997) 109–133
21. Horvath, T., Wrobel, S., Bohnebeck, U.: Relational instance-based learning with lists and terms. Machine Learning **43**(1/2) (2001) 53–80
22. Ramon, J., Bruynooghe, M.: A polynomial time computable metric between point sets. Acta Informatica **37**(10) (2001) 765–780
23. Srinivasan, A., Muggleton, S., King, R., Sternberg, M.: Mutagenesis: ILP experiments in a non-determinate biological domain. In Wrobel, S., ed.: Proceedings of the 4th International Workshop on Inductive Logic Programming. Volume 237. (1994) 217–232
24. Helma, C., King, R.D., Kramer, S., Srinivasan, A.: The predictive toxicology challenge 2000–2001. Bioinformatics **17** (2001) 107–108

Combining Ring Extensions and Canonical Form Pruning

Christian Borgelt

European Center for Soft Computing
c/ Gonzalo Gutiérrez Quirós s/n, 33600 Mieres, Spain
christian.borgelt@softcomputing.es

Abstract. A common problem in frequent graph mining is the size of the output, which can easily exceed the size of the database to analyze. In the application area of molecular fragment mining a promising approach to tackle this problem is to treat certain substructures as a unit. Among such structures, rings are most prominent, and by requiring that either a ring is present as a whole in a fragment, or not at all, the size of the output can be reduced considerably. In this paper I present two ways to combine such ring mining with canonical form pruning.

1 Introduction

In recent years frequent graph mining has become an area of very active research. Given a database of (attributed) graphs, one tries to find all subgraphs that appear with a user-specified minimum frequency. Some algorithms for this task rely on principles from inductive logic programming [5]. However, the vast majority transfers techniques from frequent item set mining. Examples are MolFea [9], FSG [10], MoSS/MoFa [1], gSpan [12], CloseGraph [13], FFSM [7], and Gaston [11]. A related, but slightly different approach is used in Subdue [4].

The basic idea is to grow subgraphs into the graphs of the database, adding an edge and maybe a node in each step, counting the number of graphs containing each grown subgraph, and eliminating infrequent subgraphs. Unfortunately, with this method the same subgraph can be constructed in several ways, adding its nodes and edges in different orders. The predominant method to avoid the ensuing redundant search is to define a canonical form of a graph that uniquely identifies it up to automorphisms: together with a specific way of growing the subgraphs it enables us to determine whether a given subgraph can be pruned from the search tree (see, for example, [3] for a family of such canonical forms).

Another common problem in frequent graph mining is the size of the output, which can easily exceed the size of the database to analyze. To tackle this problem, it can be useful to restrict the output to subgraphs with certain meaningful properties. In the application area of molecular fragment mining, for example, treating rings as units—that is, requiring that either a ring is present as a whole, or not at all—not only reduces the size of the output, but also improves its interpretability and speeds up the mining process considerably [6].

However, [6] employed a repository of reported fragments in order to suppress duplicate fragments. The canonical form pruning approach is clearly preferable, in particular, because it makes the method substantially simpler to parallelize. In this paper I present two ways in which ring mining—that is, using extensions by full rings instead of extensions by individual ring edges—can be combined with canonical form pruning: a filter approach and a reordering approach.

2 A Filter Approach to Ring Mining

Ring Preprocessing. Ring mining requires preprocessing the rings in the graphs (here usually: molecules) to analyze: given a user-specified range of ring sizes, all rings within this size range are marked in the graphs [6]. Technically, there are two parts: a marker in the edge attribute, fundamentally distinguishing ring edges from non-ring edges, and a set of flags identifying the different rings an edge is contained in. (Note that an edge can be part of several rings.)

Filtering Open Rings. If we require the output to have only complete rings, we have to identify and remove fragments with ring edges that do not belong to any complete ring. Since ring edges have been marked in the preprocessing, we know which edges are ring edges. Using the same procedure as for the preprocessing, we mark rings in the fragment, but only set the ring flags, while the edge type marker is kept. Afterwards we can find edges that belong to incomplete rings by simply checking whether a ring edge (as identified by its type marker) did not receive any ring flags. If there exists such an edge, the fragment is discarded.

Filtering Unclosable Rings. Canonical form pruning allows to restrict the possible extensions of a fragment [3]. For the canonical forms of gSpan [12] and MoSS/MoFa [1] these are so-called *rightmost extensions* and *maximum source extensions*, respectively. Their effect is that due to previous extensions certain nodes in the graph become unextendable, that is, no new edge may be attached to them. This can easily be exploited to prune the search. Obviously, a necessary (though not sufficient) condition for all rings being closed is that every node has either zero or at least two incident ring edges. If there is a node with only one incident ring edge, this edge must be part of an incomplete ring. Hence we can filter fragments in the search by checking whether any non-extendable node has only one incident ring edge. If this is the case, the fragment can be discarded.

Merging Ring Extensions. The two filtering methods described above work on individual edges and hence they cannot always detect if an extension by a ring edge only leads to fragments with complete rings that are infrequent. Consider, for example, a database with two molecules: one is a ring with 5 carbon atoms, the other a ring with 6 carbon atoms (all bonds are single). If we want to find fragments that are part of both molecules, we need not consider any extension by a ring bond, since there is no common fragment with a complete ring. However, any single ring bond is contained in both fragments and thus it is, in itself, a frequent fragment. Therefore, by working on individual bonds, we only recognize after constructing the full 5-ring that there is no common fragment.

To avoid such redundant search, we add all edges of a ring that an extension edge is contained in, thus distinguishing extensions that start with the same single edge, but lead into rings of different size or different composition. Then we determine the support and prune infrequent extensions, and finally we trim and merge ring extensions that share the same initial edge. Note that the resulting situation differs considerably from direct extensions by individual edges. In the first place, all extensions by ring edges, which are frequent as ring edges, but become infrequent when completed into rings (and thus cannot produce any output), have been removed. In addition, for the remaining edges all embeddings (that is, occurrences of the fragment in the graph database), which lead to infrequent fragments once rings are completed, have been eliminated.

3 A Reordering Approach to Ring Mining

Even with merging ring extensions the search is still based on an edge-by-edge scheme and thus fragments grow fairly slowly. It would be better if we could add complete rings in one step, thus adding several edges to the fragment. Unfortunately such an approach interferes with canonical form pruning, as is discussed below. However, the problems can be solved, although up to now I only achieved a solution for the breadth-first search canonical form of MoSS/MoFa [3].

Ring Preprocessing. Although for the filtering approach it is sufficient to mark rings in the user-specified size range, the reordering approach also needs *pseudo-rings*, as I call them, to be marked. These are rings of smaller size than the user specified, which consist only of edges that are part of rings within the user-specified size range. As an example consider the molecule shown on the left of Figure 3: if only rings with 5 and 6 bonds are marked, this molecule contains a pseudo-ring of size 3 comprising the atoms labeled 1, 3, and 4.

Ring Extensions and Canonical Forms. If we add complete rings in one step and want to use canonical form pruning, we have to order the new edges in such a way that the result is (as far as possible) in canonical form. This can easily be achieved with a recursive procedure similar to the canonical form test. However, this can have the effect that—due to asymmetries in the ring—we commit to a numbering of the nodes that prevents us from finding certain fragments. The reason is that in order to be in canonical form an (asymmetric) ring may need its nodes to be numbered differently depending on whether it is considered purely (i.e. only ring edges are present) or whether it has branches attached to it.

As an example consider the molecules shown in Figure 1 and assume the orderings `N` $\prec$ `O` $\prec$ `C` and `=` $\prec$ `-`. The nodes of the pure ring can be numbered in two ways in a breadth-first manner starting at the nitrogen atom (as shown in the top part of Figure 1). Of course, they lead to different code words, which are shown in the middle. Since single bonds precede double bonds, the upper code word is smaller and thus the left fragment is in canonical form. If, however, we consider the ring with an oxygen atom attached (as shown in the lower part of Figure 1), the alternative node numbering yields the canonical form. The reason

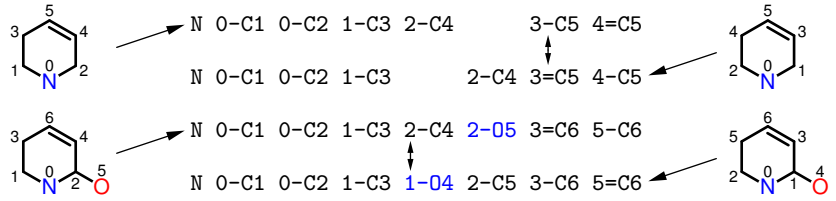


Fig. 1. Attaching a branch to a ring can change the ordering for the canonical form.

is that the attached bond to the oxygen atom has to be inserted into rather than appended to the code word. As a consequence we cannot simply commit to the numbering of the nodes as it is prescribed by the canonical form of a ring, since this numbering may change when branches or other rings are added.

A second, even nastier problem is posed by connected or even nested rings: if two ring extensions follow each other, their edges may have to be "spliced" to construct a proper code word. In doing so, one has to take care that edges can be reordered in a sufficiently flexible way, so that no fragments get lost due to the canonical form pruning. On the other hand, one also has to make sure that no duplicates result, which cannot be detected with a canonical form test. This is fairly difficult to achieve, in particular if the fragment contains branching points of equivalent ring edges, that is, ring edges that all start at the same node and have the same edge attribute and the same destination node attribute.

Canonical Form Pruning for Ring Extensions. A strategy to overcome the above problems consists in keeping (and thus also extending) fragments that are not in canonical form, but that could become canonical once branches are added. Which non-canonical fragments to keep is determined as follows: adding all edges of a ring can be seen as adding its initial edge as in an edge-by-edge procedure, and some additional edges, the positions of which are not yet fixed. Hence we split the code word into two parts: a fixed part, which would also be built by an edge-by-edge procedure, and a volatile part comprising the additional bonds. The fixed part is the prefix of the code word up to (and including) the last edge added in an edge-by-edge manner, and the volatile part is the suffix of the code word after this edge. If the current code word deviates from the canonical code word in the fixed part, the fragment is pruned, otherwise it is kept.

As an example consider the search tree shown in Figure 2. The search starts at the nitrogen atom and constructs the ring with both possible numberings of the nodes. The form in the left branch is canonic (indicated by a solid box), so it is kept. In the form in the right branch only the first ring bond (from the nitrogen atom to the right carbon atom) is fixed, every other bond is volatile. Since the code word for this fragment deviates from the canonical one only at the 5th bond (see Figure 1), we may not discard it. However, we mark it as non-canonical (indicated by a dashed box), so that it is not reported.

On the next level, adding the bond to the oxygen atom in the two possible places yields, for each branch, one canonical (solid box) and one non-canonical

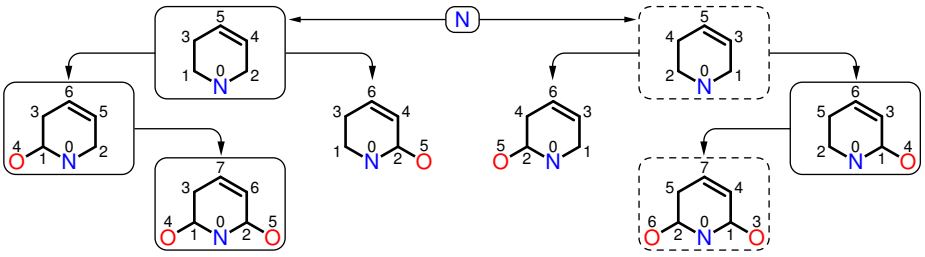


Fig. 2. Search tree for an almost symmetric ring with two identical branches.

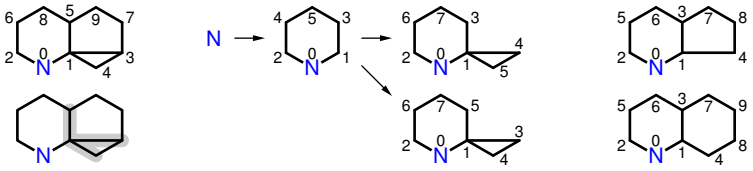


Fig. 3. Splicing equivalent bonds of ring extensions to achieve canonical form.

fragment (no box). The non-canonical fragments both differ in the fixed part, which now consists of the first three bonds[1], and hence they are both pruned. Extending the canonical fragments adds the other bond to an oxygen atom, thus making the fragment symmetric again (with the exception of the double bond). Hence the left fragment is in canonical form. The fragment in the right branch, however, is not canonical. Nevertheless it has to be kept: the first four bonds are fixed, but it deviates from the canonical code word only in the 7th bond.

In order to handle connected and nested rings, we have to consider how the edges of a ring extension may be "spliced" with edges already in the fragment. The problem here are branching points, where several equivalent edges start (i.e. edges with the same attribute and same attribute of the destination node). As an example consider the molecule on the left of Figure 3, the relevant equivalent edges of which are highlighted. Part of an example search for this fragment (shown in the middle) starts from the nitrogen atom, adds a 6-bond ring and then a 3-bond ring. If we only allowed, for example, shifts of new bonds to the left up to equivalent bonds, we can only reach the upper form. However, the canonical form is the lower one. On the other hand, we may not reorder equivalent edges freely, as this would interfere with keeping certain non-canonical fragments: the fragments in the first level of the search tree in Figure 2 differ only in the order of the two equivalent bonds starting at the nitrogen atom. If adding another ring to this atom could change this order, duplicate fragments could result.

[1] Note how adding the bond to the oxygen atom turned one of the volatile ring bonds into a fixed one: everything preceding the bond characterizing an extension—whether it is a single non-ring bond or the first bond of a ring extension—becomes fixed.

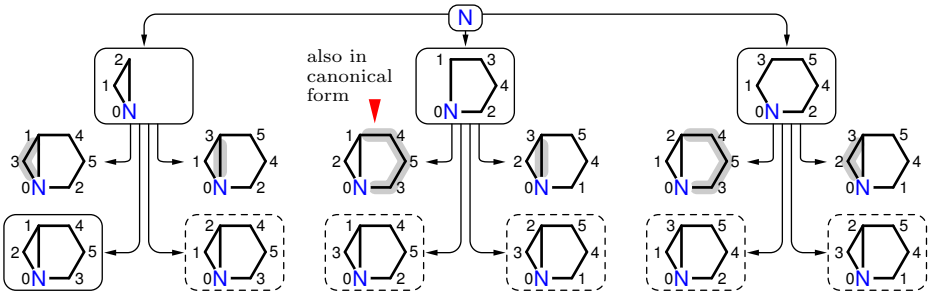


Fig. 4. Search tree for nested rings of different size (with ring order pruning).

As a consequence, the splicing rule for equivalent edges must be as follows: the order of the equivalent edges already in the fragment must be maintained, and the order of the equivalent new edges must be maintained (to avoid "flipping" rings). In addition, no new edge must be shifted into the fixed part. However, any sequence satisfying these conditions is acceptable and must be considered. In other words: the two sequences of equivalent edges—one from the existing fragment and one from the added ring—may be merged in a zipper-like manner, selecting the next edge from either list, but maintaining the order in each list.

This also explains why we need pseudo-rings: without them it would be impossible in some cases to achieve canonical form. If, in the example, we could only add the 5-ring and the 6-ring, but not the 3-ring (see the right of Figure 3), the upward bond from the atom numbered 1 would always precede at least one of the other two bonds that are equivalent to it. However, in the canonical form (shown at the top left of Figure 3) the upward bond succeeds both other bonds.

Unfortunately, the same fragment can now be reached in the same form in different ways. To see this, consider the search tree shown in Figure 4: the same fragment results in six different forms, each of which (including the canonical form) occurs twice. Hence we need an additional test to augment canonical form pruning. The idea underlying this test is that the canonical form not only fixes an order of the bonds, but also an order of the rings: order the rings by the first edge in the canonical form that is contained in them. Break ties by considering the second, third etc. edges in the canonical form the rings contain.

This ring order is exploited as follows: we mark all edges in the fixed part of the fragment as well as all new edges. Then we traverse the unmarked ring edges. For each of these edges we eliminate all rings it is contained in (by removing ring flags, which are set in the fragment in the same way as in the ring preprocessing). If by this procedure all ring flags of a marked edge (fixed edge or new edge) are removed, this marked edge is part of a ring preceding the newly added one (w.r.t. the ring order defined above). Therefore it had to be added before the current ring. If, however, no marked edge loses all its ring flags, there exists a ring that succeeds the currently added one. This ring should rather be added later and thus the fragment is discarded, even if it is in canonical form.

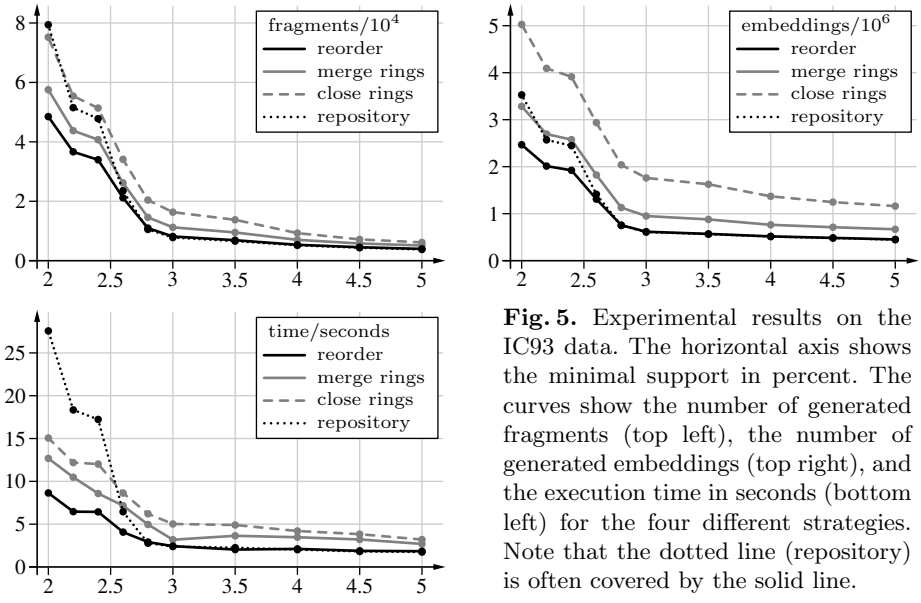


Fig. 5. Experimental results on the IC93 data. The horizontal axis shows the minimal support in percent. The curves show the number of generated fragments (top left), the number of generated embeddings (top right), and the execution time in seconds (bottom left) for the four different strategies. Note that the dotted line (repository) is often covered by the solid line.

As an example consider again Figure 4. The canonical form (fragment at bottom left) shows that the 6-bond ring succeeds both the 3-bond ring and the 5-bond ring, as it does not contain the first bond. The second bond shows that the 3-bond ring precedes the 5-bond ring. For the second fragment in canonical form (see arrow) the test, as outlined above, fails, using any of the highlighted edges. However, for the leftmost fragment in the bottom row the ring flag removal always leaves a marked edge flagless and thus it is accepted.

4 Experiments

I incorporated the described methods into the MoSS program.[2] As a test dataset I used the well-known subset of the *Index Chemicus* 1993 [8]. The results are shown in Figure 5: each curve corresponds to one of the approaches, which were executed with (full) perfect extension pruning [2]. The black dotted line refers to the repository-based approach, the solid black line to the reordering approach. The filter approach I used in two flavors: with ring extensions and consecutive merging (solid grey line) and using only open and unclosable ring filtering (dashed grey line). These results show that the somewhat complicated treatment of ring extensions in order to combine them with canonical form pruning clearly pays off, as the reordering method hugely outperforms the filter approaches.

[2] MoSS is available for download under the Gnu Lesser (Library) Public License at `http://fuzzy.cs.uni-magdeburg.de/~borgelt/moss.html`.
The test system was a Lenovo Thinkpad X60s (Intel Core Duo@1.67GHz, 1GB) with S.u.S.E. Linux 10.0, IBM's Jikes compiler 1.22, and Sun's Java 2 1.5.0_07.

5 Conclusions

In this paper I developed two methods to combine canonical form pruning with ring extensions, which are very useful for molecular fragment mining. The first method is based on a filter approach, which exploits the requirement for closed rings to restrict the output and prune the search. This approach is simple and straightforward, but not competitive in terms of running time or memory usage as the experiments show. Therefore it is indeed worthwhile to invest into the more complicated reordering method, which enables us to do full ring extensions, thus considerably reducing the height of the search tree and its number of nodes.

References

1. C. Borgelt and M.R. Berthold. Mining Molecular Fragments: Finding Relevant Substructures of Molecules. *Proc. IEEE Int. Conf. on Data Mining*, 51–58. IEEE Press, Piscataway, NJ, USA 2002
2. C. Borgelt, T. Meinl, and M.R. Berthold. Advanced Pruning Strategies to Speed Up Mining Closed Molecular Fragments. *Proc. IEEE Conf. on Systems, Man and Cybernetics*, CD-ROM. IEEE Press, Piscataway, NJ, USA 2004
3. C. Borgelt. On Canonical Forms for Frequent Graph Mining. *Proc. 3rd Int. Workshop on Mining Graphs, Trees and Sequences*, 1–12. ECML/PKDD 2005 Organization Committee, Porto, Portugal 2005
4. D.J. Cook and L.B. Holder. Graph-Based Data Mining. IEEE Trans. on Intelligent Systems 15(2):32–41. IEEE Press, Piscataway, NJ, USA 2000
5. P.W. Finn, S. Muggleton, D. Page, and A. Srinivasan. Pharmacore Discovery Using the Inductive Logic Programming System PROGOL. *Machine Learning*, 30(2–3):241–270. Kluwer, Amsterdam, Netherlands 1998
6. H. Hofer, C. Borgelt, and M.R. Berthold. Large Scale Mining of Molecular Fragments with Wildcards. *Intelligent Data Analysis* 8:495–504. IOS Press, Amsterdam, Netherlands 2004
7. J. Huan, W. Wang, and J. Prins. Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism. *Proc. 3rd IEEE Int. Conf. on Data Mining*, 549–552. IEEE Press, Piscataway, NJ, USA 2003
8. *Index Chemicus — Subset from 1993*. Institute of Scientific Information, Inc. (ISI). Thomson Scientific, Philadelphia, PA, USA 1993
`http://www.thomsonscientific.com/products/indexchemicus/`
9. S. Kramer, L. de Raedt, and C. Helma. Molecular Feature Mining in HIV Data. *Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 136–143. ACM Press, New York, NY, USA 2001
10. M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. *Proc. 1st IEEE Int. Conf. on Data Mining*, 313–320. IEEE Press, Piscataway, NJ, USA 2001
11. S. Nijssen and J.N. Kok. A Quickstart in Frequent Structure Mining Can Make a Difference. *Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 647–652. ACM Press, New York, NY, USA 2004
12. X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. *Proc. 2nd IEEE Int. Conf. on Data Mining*, 721–724. IEEE Press, Piscataway, NJ, USA 2002
13. X. Yan and J. Han. Closegraph: Mining Closed Frequent Graph Patterns. *Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 286–295. ACM Press, New York, NY, USA 2003

Structured Kernels for Automatic Detection of Protein Active Sites

Elisa Cilia[1], Alessandro Moschitti[1], Sergio Ammendola[2], and Roberto Basili[1]

[1] Department of Computer Science, System and Production
University of Rome, Tor Vergata,
Via Della Ricerca Scientifica s.n.c., 00133, Roma, ITALY
elisa.cilia@gmail.com
{moschitti,basili}@info.uniroma2.it
[2] Ambiotec sas
SME
field: biotechnology
Via F. Acqua Mariana 125, 00040, Roma, ITALY
PI 07635910636
sergio.ammendola@fastwebnet.it

Abstract. In this paper, we design novel models based on Support Vector Machines and Kernel Methods for the automatic protein active site classification. We devise innovative attribute-value and tree substructure representations derived from biological and spatial information of proteins. We experimented such models with the Protein Data Bank adequately pre-processed to make explicit the active site information. Our results show that structural kernels used in combination with polynomial kernels can be effectively applied to discriminate an active site from other regions of a protein. Such finding is very important since it firstly shows the successful identification of catalytic sites of a very large family of catalytic proteins belonging to a broad classes of enzymes.

1 Introduction

Recent research in Bioinformatics has been devoted to the production and understanding of genomic data. One important step in this direction is the study of the relation between molecular structures and their functions, which in turn depends on the discovering of the protein active sites. As there is a large number of synthesized proteins which have no associated function yet, i.e. whose function remains unknown, automatic approaches for active site detection are critical.

Currently, the general strategy used to identify a protein active site involves the expertise of researchers and biologists accumulated in years of study on the target protein as for example in [1]. This manual approach is conducted essentially using homology based strategies, i.e. inferring the function of a new protein based on a close similarity to already annotated proteins. Sometimes proteins with the same overall tertiary structure can have different active sites, i.e. different functions and proteins with different overall tertiary structure can

show the same function and similar active sites. In these cases homology based approaches are inadequate. Moreover, there is no general automated approach to protein active site detection, although it is evident its usefulness to restrict the number of candidate sites and also to automatically learn rules characterizing an active site [2].

In this paper we define the problem of determining protein active sites in terms of a classification problem. We modeled protein active site based on both attribute/value and structural representations. The former representation is a set of standard linear features whereas the latter is constituted by tree structures extracted from graphs associated with proteins or their candidate sites. The graph nodes (or vertexes) represent amino acids (or better residues) and edges represent distances in the three-dimensional space between these residues.

We applied these representations to SVMs using polynomial kernels, tree kernels and some combinations of them. To experimentally evaluate our approach, we created a data set, using the protein structures retrieved from the *Protein Data Bank* (PDB) [3] maintained by the *Research Collaboratory for Structural Bioinformatics* (RCBS) at http://www.rcbs.org. The combined kernels show the highest F_1 measure, i.e. 68%, in the detection of active sites. This is an important and promising result considering that the baseline based on a random selection of active sites has an upperbound of only 2%.

In the remainder of this article Section 2 describes the faced problem. Section 3 describes the proposed linear and structural features. Section 4 describes the experimental evaluation and reports the results of the classification experiments. Finally, in Section 5, we summarize the results of the previous sections and propose other interesting future research lines.

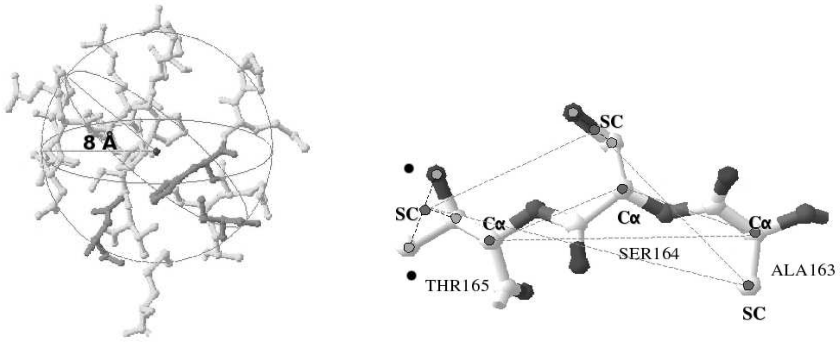
2 Protein active site classification

An active site in a protein is a topological region which defines the protein function, in other words it is a functional domain in the protein three-dimensional structure (see also [2]). In a cell there are many types of proteins which carry out different functions. The enzymes are those proteins able to accelerate chemical processes inside a cell. This type of proteins are distinguished from structural and supplying proteins for their catalytical action on the large part of molecules constituting the living world. We limit our research to a particular class of enzymes, the hydrolases.

Hydrolases are maybe the most studied and known type of enzymes. They catalyze hydrolysis reactions, generically consisting in the cleavage of a biochemical compound thanks to the addition of a water molecule (H_2O). The characteristic of some hydrolases to catalyze reactions in the presence of a water molecule motivates our model: as an hydrolase active site, we choose a sphere in a three-dimensional space centered in the coordinates of the oxygen atom of a water molecule. This sphere includes a portion of the protein within its volume, that is a number of amino acids which could reciprocally interact with other amino acids in the surrounding space, or with water molecules. In this first analysis, we

consider a sphere with a radius of 8 Å, which is the maximum distance needed for the water-residue interaction.

Figure 1(a), shows the active site of 1A2O protein structure and its representation according to our model. The protein residues are colored in light gray whereas the particular catalytic residues are in dark gray. The center of the sphere is the black colored oxygen atom of a water molecule.



(a) (b)

Fig. 1. (a) A sphere (positive example). (b) Distances.

2.1 The computational model

As stated in the previous section, we defined the functional site identification as a classification problem, where the objects we want to classify are protein active sites. We represent the portion of the protein contained in a spherical three-dimensional region with a completely connected graph. Each vertex of this graph is a residue and each edge represents the distance in the three-dimensional space between a pair of vertexes.

Every amino acid is represented by two points in the three-dimensional space: one which represents an amino acid main-chain (the α-carbon atom of the amino acid, Cα) and one which represents an amino acid side-chain (the centroid between the coordinates of the atoms belonging to the amino acid side-chain, SC) (see Figure 1(b)). The same kind of approximation has been described in [4] because it seems to provide a good balance between fuzziness and specificity in these kind of applications.

In Figure 1(b) the three-dimensional SC-SC distances and Cα-Cα distances are indicated between the represented chain of three residues. An object (modeled by a graph centered on a water molecule) can be classified as being an active site or not with a binary classifier. Thus, we consider as a positive example, a graph whose set of vertex includes all the catalytic amino acids and as a negative example a graph which contain no catalytic amino acid. Moreover, to reduce the

task complexity, we extract, from the initial completely connected graph, some spanning substructures which preserve the edges within the maximum interaction distance of 5 Åbetween the side-chains of the residues.

The next section shows how the above representation model can be used along with Support Vector Machines to design an automatic active site classifier.

3 Automatic classification of active sites

Previous section has shown that the active site representation is based on graphs. To design the computational model of these latter, we have two possibilities: (1) we extract scalar features able to capture the most important properties of the graph and (2) we can use graph based kernels [5] in kernel-based machines such as Support Vector Machines [6]. Point (2) often leads to high computational complexity. We approached such problem by extracting a tree forest from the target graph and applying efficient tree kernels [7].

3.1 Scalar features

Scalar features refer to typical chemical values of the molecules described in the target graph. We defined 5 different types of such features (see Table 1):

The first class of linear features (C1) encodes chemical and physical properties of the graph. This class represents properties such as hydrophobicity, polarity, polarizability and Van der Waals volume of the amino acids composing the sphere. The encoding is the same used in [8] where the features were used to classify the function of proteins.

The second class of linear features (C2) encodes the amino acid composition of a spherical region. There is a feature associated with every labeled vertex (amino acid) in the graph, weighted with the inverse of the distance from the oxygen atom of the water molecule which is the center of the sphere. This group of features emphasizes the importance of the interaction distance of a residue with respect to a water molecule.

The third class of features (C3) represents charge or neutrality of a spherical region. This is measured by counting the number of positively or negatively charged amino acids.

Another group of linear features (C5) encodes the quantity of water in a sphere. This is measured by counting the number of water molecules within the sphere radius. This group of features is motivated by the fact that biologists observed that an active site is usually located in a hydrophobic core of the protein while on the surface the quantity of water is higher and the residues exposed to the solvent are not hydrophobic.

Finally, the last class of linear features (C6) is the one which measures the atomic density of the sphere calculated as the total number of atoms in the sphere.

It should be noted that (a) the last two classes of linear features are made discrete using a different number of value intervals. A feature is associated with

Table 1. Representation: feature classes

Linear Features	**Description**
1st Class	Physical and chemical properties (amino acid attributes)
2nd Class	Amino acidic Composition
3rd Class	Charge/Neutrality
5th Class	Water molecule quantity
6th Class	Atomic density
Structural Features	**Description**
4th Class	Tree substructures from tertiary structure

an example if the measured value of a certain property falls in the correspondent range. (b) These features are often used to describe protein structures in similar tasks of Bioinformatics [8] and to develop software for protein structure prediction like Modeler 7v7.

3.2 Structural features

We designed a class of structural features to encode the three-dimensional structure (tertiary structure) or better, the spatial configuration characterizing a spherical region, i.e. the set of amino acids composing it with their 3D distances. As previously mentioned this representation results in a completely connected graph since every vertex is connected to any other vertex in the sphere graph through an edge labeled with the 3D distance of the pair.

Starting from this completely connected graph, we extract some tree substructures using heuristics: for example, the one which preserves the maximum interaction distances to 5 Åbetween the side-chains of the residues and a minimum spanning tree algorithm. Such heuristics is motivated by the observation that to perform the catalytic function it is necessary that the side-chains of the catalytic residues can interact with each other and with the substrate. The maximum interaction distance between atoms in different residue side-chains is usually of about 3-4 Å. We chose a cut-off distance of 5 Åto take into consideration our approximation in the representation of residues (Figure 1(b)).

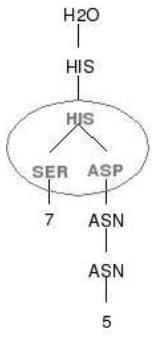


Fig. 2. Graphical representation of a tree of a sphere

The applied heuristics lead possibly to the separation in disconnected components of the initial graph. From each of these components, using the Prim algorithm [9], we extract the spanning tree which minimizes the cost function $c(T)$, i.e. the interaction distances d_{xy} between the side-chains of the residues x and y. Note that as some graphs contain more than one connected component,

the Prim algorithm is applied to each of them. This leads to the extraction of a tree forest.

We add the water molecule (center of the sphere) as root node to the obtained spanning tree. In Figure 2, we show a tree which can represent the spherical region in Figure 1(a). In bold light gray, we highlight the nodes which represent catalytic amino acids.

The tree substructures generated for each example constitute the features analyzed by our tree kernel function. If two examples are described by two tree forests, we can use as a kernel function the summation of a tree kernels applied to all possible pairs coming from such forests.

4 Experiments

In the subsequent subsections, we describe our classification experiments carried out on the data set that we generated from the Protein Data Bank.

4.1 Experimental set-up

The evaluations were carried out using the SVM-light-TK software [10] (available at http://ai-nlp.info.uniroma2.it/moschitti/) which encodes tree kernels in SVM-light [11]. We used the polynomial kernels for the linear features and tree kernels for the structural feature processing. More precisely, we used the SST and the PT kernels described in [7] on a simple tree, i.e. the main tertiary structure[3], or on a tree forest (see Section 3.2). The former kernels are indicated with SST_T and PT_T whereas the latter are called SST_F and PT_F. The kernel for tree forest is simply the summation of all possible pairs of trees contained in two examples.

We experimented our models with the protein structures downloaded from the Protein Data Bank (PDB). We adequately pre-processed PDB files to obtain all the information of interest for this task. In particular, we created a data set of 14,688 examples from 48 hydrolases from the PDB structures. The data set is composed of 171 positive examples and 14,571 negative examples, which means a $\frac{1}{125}$ ratio between positive and negative examples.

The results were evaluated by applying a 5-fold cross validation[4] on this data set measuring the performance with the F_1 measure [5].

A noticeable attention was devoted to parameterization (cost factor, decay factor, etc.)

[3] The single tree structure is the most relevant one in the forest, that is, the tree which contains at least a catalytic amino acid and the two nearest residue side-chains of the sphere

[4] We separated the data set into five parts, each one composed of examples belonging to a set of nine or ten protein structures randomly assigned to this set.

[5] F_1 assigns equal importance to Precision P and Recall R i.e. $F_1 = \frac{2P \cdot R}{P+R}$

Table 2. (a) Linear features performance. (b) Combined kernel performance.

(a)

Linear	**Precision**	**Recall**	F_1
C1	5.5%	66.7%	**10.2%**
C2	55.9%	63.3%	**59.4%**
C3	20%	3.3%	**5.7%**
C5	2.2%	30%	**4.1%**
C6	5.5%	13.3%	**7.8%**

(b)

	Precision	**Recall**	F_1 $\pm Std.Dev.$
L	62.3%	55.4%	**56.2%** $\pm$6.8
SST_F	66.2%	31.8%	**39.9%** $\pm$13.7
L+SST_F	82.9%	58.6%	**68.3%** $\pm$14.5

4.2 Experiment results

Table 2(a) reports the results on the 5 types of linear features using the polynomial kernel (degree 3). These results are only indicative as we did not run a cross validation procedure. We note that most linear features cannot discriminate between active and non-active site. Only, the second class, which encodes the structural information, shows a meaningful F_1. The general low results of linear features is caused by the remarkable complexity of the task as suggested by the F_1 upperbound of the random selection, i.e. $\simeq 1.6\%$.

In order to boost the classification performance, we experimented with the structural kernels. Table 2(b) summarizes the cross validation results: Row 2 reports the results with polynomial kernels on all the linear features (L), Row 3 shows the outcomes of the SST kernels on the tree forest (SST_F) and Row 4 illustrates the performance of the polynomial kernel summed to the SST kernel on the tree forest (L+SST_F). The $\pm$ sign precedes the standard deviation evaluated on the 5 folds. It is worth to note that the F_1 obtained with the linear features (56.24%) improves by 12 absolute points if we use the combined model (L+SST_F), i.e. 68%.

We also experimented different variants of Tree Kernels, i.e. based on PTs. The results of the cross validation experiments are summarized in Table 3: Row 2 reports the results with polynomial kernel plus SST_F (applied to linear features and a forest structure), Row 3 reports the cross validation results of polynomial kernel plus SST_T (applied to linear features and a tree structure) and finally Row 4 illustrates the performance of the additive combination of polynomial with the PT kernel (PT_T) (on linear features and a tree structure).

L+TK	**Precision**	**Recall**	F_1 $\pm Std.Dev.$
SST_F	82.9%	58.6%	**68.3%** $\pm$14.5
SST_T	79.7%	51.7%	**62.3%** $\pm$10.4
PT_T	80.4%	41.2%	**54.4%** $\pm$9.1

Table 3. Tree kernel impact

The results show that the highest F_1 measure can be achieved with the SST_F but quite similar performance can be obtained representing examples with only a tree structure in the forest, i.e. SST_T. In contrast to our expectations the PT kernel, which may be considered the one most suitable for this task, shows the lowest F_1. The most plausible explanation is the highest complexity on deriving its correct parameterization.

Overall, the very good F_1 of our best model suggests that our classification system can be a useful tool to help the biology researcher to study the protein functions.

5 Conclusions

In this paper, we have studied the problem of the identification of protein functional sites. We have defined a novel computational representation based on biological and spatial considerations and several classes of linear and structural features.

The experiments with SVMs using polynomial and tree kernels and their combinations show that the highest F_1, i.e. 68%, is achieved with the combined model. Such finding is very important since it firstly shows the successful identification of catalytic sites of a very large family of catalytic proteins belonging to a broad classes of enzymes. Moreover, our work highlights the importance of structural information in the detection of protein active sites. This result motivates the need of structural representations which we efficiently modeled by means of tree kernels.

References

1. Cilia, E., Fabbri, A., Uriani, M., Scialdone, G.G., Ammendola, S.: The signature amidase from sulfolobus solfataricus belongs to the cx3c subgroup of enzymes cleaving both amides and nitriles: Ser195 and cys145 are predicted to be the active sites nucleophiles. The FEBS Journal **272** (2005) 4716–4724
2. Tramontano, A.: The ten most wanted solutions in Protein Bioinformatics. Chapman & Hall/CRC Mathematical Biology and Medicine Series (2005)
3. Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., Bourne, P.E.: The protein data bank. Nucleic Acids Res **28**(1) (2000) 235–242
4. Meng, E.C., Polacco, B.J., Babbitt, P.C.: Superfamily active site templates. PROTEINS: Structure, Function, and Bioinformatics **55** (2004) 962–976
5. Gärtner, T.: A survey of kernels for structured data. Multi Relational Data Mining (MRDM) **5** (2003) 49–58
6. Vapnik, V.: The Nature of Statistical Learning Theory. Springer (1995)
7. Moschitti, A.: Efficient convolution kernels for dependency and constituent syntactic trees. In: Proceedings of The 17th European Conference on Machine Learning, Berlin, Germany, 2006, Berlin, Germany (2006)
8. Borgwardt, K.: Graph-based Functional Classification of Proteins using Kernel Methods. Ludwig Maximilians University of Monaco (2004)
9. Prim, R.C.: Shortest connection networks and some generalizations. Bell Syst. Tech. Journal **36** (1957) 1389–1401
10. Moschitti, A.: A study on convolution kernel for shallow semantic parsing. In: Proceedings of the 42th Conference on Association for Computational Linguistic (ACL-2004), Barcelona, Spain (2004)
11. Joachims, T.: Making large-scale svm learning practical. In: Advances in Kernel Methods - Support Vector Learning, B. Schölkopf, C. Burges and A. Smola editors (1999)

Learning Structured Outputs via Kernel Dependency Estimation and Stochastic Grammars

Fabrizio Costa[1], Andrea Passerini[1], and Paolo Frasconi[1]

Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze

Abstract. We focus on graph-valued outputs in supervised learning and propose a novel solution to the pre-image problem in the kernel dependency estimation framework. Output structures are generated by a stochastic grammar and the output feature space is directly associated with the set of productions for the grammar. The regression estimation step learns to map input examples into a feature vector that counts the number of applications of each production rule. A max-propagation algorithm finally builds the predicted output according to the normalized counts. We test our method on a ambiguous context free grammar (CFG) parse tree reconstruction problem. We show on an artificial dataset that mimics the prepositional attachment problem how learning the number of applications of each production rule on a per example base allows CFG parser to better tackle ambiguity issues.

1 Introduction

A structured output prediction problem can be formulated as a supervised learning problem in which the output or target space is not restricted in any way and can be any set (e.g. a set of graphs or sequences). In this way, several predictions can be made collectively on the same input instance, still maintaining that instances are sampled independently. Formally, we denote by $\mathcal{X}$ and $\mathcal{Y}$ the input and output spaces, respectively. Examples are in the form $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ and are sampled identically and independently from a fixed and unknown distribution p. The cost associated with prediction errors is measured by a loss function $V : \mathcal{Y} \times \mathcal{Y} \mapsto \boldsymbol{R}$. Learning consists of finding a function $f : \mathcal{X} \mapsto \mathcal{Y}$ in a given hypothesis space such that the average loss $V(y, f(x))$ is minimized.

The above setting can be useful in many real world application domains. Structural bioinformatics offers interesting examples since there are several prediction tasks where the target is about a relation between two or more residues in a protein chain. Problems of this kind include prediction of disulfide bridges, metal binding sites, beta sheet partners and contact maps. Correlation often plays a major role linking the prediction at different residues in the same chain. It is relatively safe, however, to assume that different protein chains are sampled independently. Thus, these structural bioinformatics problems fit the above

framework choosing $\mathcal{X}$ to be a set of sequences and $\mathcal{Y}$ a set of graphs (whose vertices are the elements of the input sequence). Computational linguistics offers other interesting examples of sequential translation problems like POS-tagging and named entity recognition.

Typically, structured output prediction involves searching the output space $\mathcal{Y}$, where the search can be guided by a scoring function associated with the input x and a candidate output y.

$$f(x) = \arg\max_{y \in \mathcal{Y}} S(x, y). \tag{1}$$

Exhaustive search in spaces of graphs is generally intractable and heuristic methods are necessary. Search in graph space can be implemented starting in some initial state and iteratively moving in $\mathcal{Y}$ by applying some modification operator to the current structure. In [2, 6], parse tree construction from input sequences is formulated in the context of incremental dynamic grammars and recursive neural networks were employed to learn the best operator to be applied at each search step. In some special cases it is possible to define "unimodal" functions $S(x, y)$ for which hill climbing can be shown to be complete in solving Eq. 1. In [12], a unimodal function was proposed for scoring candidate protein contact maps and a recursive neural network was trained in regression mode to predict the value $S(x, y)$ to be plugged in Eq. 1.

Weston et al. [13] and Tsochantaridis et al. [10] have proposed kernel-based formulations of the problem in Eq. 1. Here we follow-up the kernel dependency estimation (KDE) framework introduced in [13] and further specialized to the case of sequential transductions in [1].

2 Kernel Dependency Estimation

In KDE [13, 1], both the input and the output portions of the data are mapped into their feature spaces, denoted as $\mathcal{F}_\mathcal{X}$ and $\mathcal{F}_\mathcal{Y}$, respectively. As usual, these mappings can be implicitly defined via two kernel functions:

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle \tag{2}$$

$$\lambda(y, y') = \langle \psi(y), \psi(y') \rangle \tag{3}$$

where $\phi : \mathcal{X} \mapsto \mathcal{F}_\mathcal{X}$ and $\psi : \mathcal{Y} \mapsto \mathcal{F}_\mathcal{Y}$ are the input and the output feature mapping, respectively. Then every function $f : \mathcal{X} \mapsto \mathcal{Y}$ in the original domain, can be mapped to a corresponding function $F : \mathcal{F}_\mathcal{X} \mapsto \mathcal{F}_\mathcal{Y}$ in the transformed space defined as $F(\phi(x)) = \psi(f(x))$. KDE consists of two separate steps: feature estimation and pre-image calculation, as detailed below.

Feature estimation problem. This step consists of predicting the image of the target $\psi(y)$ given the input x. We need the assumption that $\mathcal{F}_\mathcal{Y}$ has finite dimension n_o (or alternatively we could apply kernel PCA to reduce its dimensionality to a finite integer).

The feature estimation problem can be conveniently represented as a vector-output regression problem where one learns a function $g : \mathcal{X} \mapsto \mathcal{F}_{\mathcal{Y}}$ from examples $\{(x_i, \psi(y_i))\}$. It can be shown that using kernel ridge regression, the n_o regression problems can be solved using a single matrix inversion, obtaining the general solution

$$g(x) = \sum_{i=1}^{m} c_i \kappa(x, x_i) \tag{4}$$

being $c_i \in \boldsymbol{R}^{n_0}$ the columns of the solution to the following linear problem:

$$C = \Psi(y)(K + \gamma m I_m)^{-1} \tag{5}$$

being K the input kernel matrix and $\Psi(y)$ the $n_o \times m$ matrix with columns $\psi(y)$. Efficient alternatives to the above approach consist of solving each of the n_o regression problems by support vector regression [11] or using maximum margin regression [8, 7].

Pre-image calculation problem. Once the feature space representation of the target has been estimated as $g(x)$, we are left with the problem of inverting the output feature mapping $\psi(y)$ in order to obtain a predicted $f(x) \in \mathcal{Y}$. The approach suggested in [13, 1] consists of searching the space $\mathcal{Y}$ for a structure whose image in $\mathcal{F}_{\mathcal{Y}}$ is close to $g(x)$:

$$f(x) = \arg\min_{y \in \mathcal{Y}} \|g(x) - \psi(y)\|^2 \tag{6}$$

When using kernel ridge regression, it can be shown that

$$\|g(x) - \psi(y)\|^2 = \lambda(y, y) - 2 \sum_{i,j=1}^{m} h_{ij} \lambda(y_i, y) \kappa(x_j, x) \tag{7}$$

being $H = \{h_{ij}\} = (K + \mu I)^{-1}$. In [1] the search problem of Eq.(6) is solved by a graph theoretical algorithm in the case of output strings and k-gram output kernels.

3 Using stochastic grammars

We propose here an alternative KDE formulation where the pre-image problem is solved via probabilistic inference in stochastic grammars. We focus on supervised problems where input instances are arbitrary objects (e.g. strings) and the associated targets y are the result of a generative mechanism described by a stochastic grammar which depends on the input instance. One example is language learning where x is a string in a finite alphabet T and y a parse tree of x (which is not necessarily unique if the grammar is ambiguous). Another example is string transduction where x and y are strings in different alphabets and y is generated by selecting suitable production rules on the basis of x. Let $\mathcal{G}(x) = \{N, T, \mathcal{S}, \Pi(x)\}$ denote the stochastic grammar associated with x, where

N is the set of nonterminal symbols, T the set of terminal symbols, $\mathcal{S}$ the set of production rules and $\Pi(x)$ the corresponding probabilities. In our notation we have made explicit the assumption that the structure $\mathcal{S}$ of the grammar is fixed and given as background knowledge to the learner, while the parameters $\Pi(x)$ depend on x and are unknown. Let $\mathcal{L}(x)$ be the language generated by $\mathcal{G}(x)$ and for $y \in \mathcal{L}(x)$ let $\Pr(y|x)$ denote the probability that $\mathcal{G}(x)$ has generated y: in order to solve the statistical learning problem outlined in the introduction, we need that $\Pr(y|x)$ be a good model for $p(y|x)$.

We link this generative process to KDE by choosing as features $\psi(y)$ a real vector from which $\Pi(x)$ can be conveniently obtained.

While in this phase there is in principle no constraint on the kind of stochastic grammar to employ (context free, context sensitive, or even more expressive!), when we finally get down to the computation of the pre-image of $\psi(y)$ we have to opt for computationally tractable methods. To this end we choose a stochastic context free grammar to model the structured output information so to exploit efficient maximum-propagation algorithms (such as Viterbi [3]) which can, in practice, run with complexity lower than $O(|x|^3)$. In this case $\psi(y)$ is a vector of frequency counts for the rules used in the parse tree y.

Formally, suppose $\mathcal{G}(x)$ is a stochastic context free grammar. Production rules $r_{k\ell}$ have the form $A_k \mapsto \alpha_\ell$ with $A_k \in N$ and $\alpha_\ell \in (N \cup T \cup \{\epsilon\})^*$. Each production rule $r_{k\ell}$ has an attached probability $\pi_{k,\ell}$ with constraints $\sum_\ell \pi_{k,\ell} = 1$ for each $k = 1 \ldots, |N|$. These probabilities are linked to the feature vector $\psi(y)$ by the softmax function:

$$\pi_{k,\ell} = \frac{e^{\psi_{k,\ell}}}{\sum_{j=1} e^{\psi_{k,j}}}. \tag{8}$$

In this way, the feature estimation step of KDE consists of solving the regression problem for a multinomial logit model, i.e. a generalized linear model [5]. Note how the expressive power of the grammar describing the output structure is greater than simple SCFG as the probabilities associated to each rule depend on the inputs x. Informally, first we learn $\psi(y)$ i.e. the frequency counts of each rule in $\mathcal{G}$ for the parse tree y then we give these estimates to a SCFG parser that computes the pre-image of $\psi(y)$ i.e. builds the actual parse tree. We call the overall procedure KDE-SCFG.

For another approach to the task of learning the structured output of a SCFG parse tree in terms of its production rules see [9] where they learn a kernel machine that discriminates among the entire space of parse trees factorized in an extended bottom-up tabular representation.

4 Experiments

4.1 The artificial task

We test the proposed method on an artificial dataset. We are interested in problems where instances' output structure cannot be well explained resorting only

to a SCFG (in this case we could use the standard parsing techniques). We simulate a problem of interest in the NLP domain known as the PP-attachment ambiguity resolution problem which is known to be context-sensitive. The task consists in deciding which of two possible structural parse tree that involve a preposition is the correct one. To clarify the issue consider the following two sentences "eat a salad with a fork" and "eat a salad with tomatoes": in the first one the propositional phrase (PP) "with a fork" specifies a characteristic of the action of eating, while in the second case it specifies a property of the salad; in the first case we have a parse tree where the PP is attached to the verb 'eat' as in: (VP (V eat) (NP a salad) (PP with a fork)) while in the second case the PP is attached to the noun 'salad' as in (VP (V eat) (NP a salad (PP with tomatoes))). A CFG that has access to the part of speech (POS) of the sentence words only has no way to discriminate between the two alternatives which are both syntactically correct: the disambiguation can happen only lexicalizing the grammar i.e. making the rules dependent on the actual words, which is a way to introduce a form of context-sensitiveness.

We simulate the PP-attachment problem using the following simple stochastic context free grammar $\mathcal{G}$:

$$\begin{array}{rr} S \rightarrow ScS|NV & w \rightarrow 5 \\ V \rightarrow wNP|vN_P & v \rightarrow 4 \\ N \rightarrow n|ncV & n \rightarrow 2|3 \\ N_P \rightarrow nP|ncVP & p \rightarrow 1 \\ P \rightarrow pn & c \rightarrow 0 \end{array}$$

where $N = \{S, V, N, N_P, P, c, n, p, v, w\}$ and $T = \{0, 1, 2, 3, 4, 5\}$ and $\{c, n, p, v, w\}$ are the POS tags (pre-terminal). The probabilities are all uniform except for the S derivation for which we set a .2 probability of deriving ScS against a .8 probability for NV.

To introduce the context-sensitiveness we collapse the POS tags 'v' and 'w' in a single tag 'x': now, given the structure of $\mathcal{G}$ and a dataset wide global estimate of Π, the SCFG parser has no deterministic information as to which expansion rule to use for the verbal phrase: the resulting grammar is ambiguous. We want to show that exploiting similarities between the inputs in sequential form we can learn the probabilities associated to the different rules used to resolve the ambiguity and inform the SCFG parser on a per example base on which rule to prefer hence obtaining a better parse tree.

4.2 Data preparation

We used Douglas Rohde's Simple Language Generator (SLG) program[1] to randomly produce sets of sentences according to $\mathcal{G}$. We then post-processed the sets

[1] http://tedlab.mit.edu/ dr/SLG/

with two strategies: in the first one (Natural) we filtered out sentences with the same sequence structure (and therefore with the same parse trees), while in the second set (Unique) we filtered out sentences with identical representation in the output feature space $\mathcal{F}_\mathcal{Y}$.

4.3 Results

We compared KDE-SCFG with a standard SCFG parser that makes use of probabilities globally estimated over the entire dataset. We employed a spectrum kernel [4] with k-mers of size 2 to 5 to compute κ. We use Collin's `evalb` program[2] to compute the bracketing F-measure and exact parse matching scores. We randomly split the dataset in two equally sized sets of 1,000 instances each, which were employed for model selection and final evaluation respectively, both performed by a 5-fold cross validation procedure. Table 1 reports micro-averaged results of the 5-fold cross validation evaluation procedure, both for the entire evaluation set and focused on short sequences (< 35 terminals) only. The results indicate that the KDE-SCFG approach outperforms the SCFG parser significantly ($p > .05$ in all pairwise comparisons).

Table 1. Comparison between standard SCFG and KDE-SCFG

Filtering	Natural		Unique	
Measure	F-score	Exact	F-score	Exact
$\text{SCFG}_{<35}$	86.4	10.3	84.7	3.1
SCFG	85.8	8.1	84.4	0.5
$\text{KDE-SCFG}_{<35}$	93.3	33.2	94.3	28.6
KDE-SCFG	91.5	26.1	89.6	4.8

5 Conclusions

We introduced a novel solution to the pre-image problem for kernel dependency estimation using an output feature space associated to the frequency of context free grammar production rules. We showed that learning their frequency on a per instance base is an effective way to approximate a specific context sensitive grammar on a simplified NLP problem significantly outperforming a standard stochastic context free parser. The key ideas introduced in this (preliminary) work can in principle be extended to more complex settings: as an example consider associating the feature space with clauses in a probabilistic inductive logic programming setting and running logical inference procedures to find an explanation (proof) for the target concept.

[2] http://nlp.cs.nyu.edu/evalb/

6 Acknowledgement

This work was in part supported by the APrIL Project under contract no. FP6-508861. The authors would like to thank Manfred Jaeger from Aalborg University for useful comments and discussions.

References

1. C. Cortes, M. Mohri, and J. Weston. A General Regression Technique for Learning Transductions. *Proceedings of the 22nd international conference on Machine learning*, pages 153–160, 2005.
2. F. Costa, V. Lombardo, P. Frasconi, and G. Soda. Wide coverage incremental parsing by learning attachment preferences. In F. Esposito, editor, *AI*IA 2001: Advances in Artificial Intelligence, 7th Congress of the Italian Association for Artificial Intelligence*, volume 2175 of *Lecture Notes in Computer Science*, pages 297–307. Springer, 2001.
3. Karim Lari and Steve J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.
4. C. Leslie, E. Eskin, and W.S. Noble. The spectrum kernel: a string kernel for svm protein classification. In *Proc. of the Pacific Symposium on Biocomputing*, pages 564–575, 2002.
5. P. McCullagh and J.A. Nelder. *Generalized Linear Models*. Chapman & Hall, 1983.
6. P. Sturt, F. Costa, V. Lombardo, and P. Frasconi. Learning first-pass attachment preferences with dynamic grammars and recursive neural networks. *Cognition*, 88(2):133–169, 2003.
7. S. Szedmak and J. Shawe-Taylor. Multiclass and Multiview Learning at One-class Complexity. Technical report, ISIS Group Electronics and Computer Science, 2005.
8. S. Szedmak, J. Shawe-Taylor, and E. Parado-Hernandez. Learning via Linear Operators: Maximum Margin Regression. Technical report, Pascal Research Reports, 2005.
9. Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. Max-margin parsing. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 1–8, Barcelona, Spain, July 2004. Association for Computational Linguistics.
10. I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large Margin Methods for Structured and Interdependent Output Variables. *The Journal of Machine Learning Research*, 6:1453–1484, 2005.
11. V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.
12. A. Vullo and P. Frasconi. A bi-recursive neural network architecture for the prediction of protein coarse contact maps. In *1st IEEE Computer Society Bioinformatics Conference (CSB'02)*, pages 187–196, Stanford, CA, 2002.
13. J. Weston, O. Chapelle, A. Elisseeff, B. Schölkopf, and V. Vapnik. Kernel dependency estimation. In *NIPS*, pages 873–880, 2002.

Distance-Based Generalisation Operators for Graphs *

Vicent Estruch, César Ferri, José Hernández-Orallo, and María José Ramírez-Quintana

DSIC, Univ. Politècnica de València , Camí de Vera s/n, 46020 València, Spain.
{vestruch,cferri,jorallo,mramirez}@dsic.upv.es

Abstract. Distances and similarity functions between structured datatypes, such as graphs, have been widely employed in machine learning since they are able to identify similar cases or prototypes from which decisions can be made. In these distance-based methods the justification of the labelling of a new case a is usually based on expressions such as "label(a)=label(b) because case a is similar to case b". However, a more meaningful pattern, such as "because case a and b have properties x and y" is usually more difficult to find since the connection of this pattern with the distance-based method might be inconsistent [4]. In this paper we study possible consistent generalisation operators for the particular case of graphs embedded in metric spaces.

1 Introduction

While in some learning problems the data can be described by a single-fixed row of nomimal or numerical attributes, in most others, such as biomedicine or web mining, a structured representation language is needed.

In general, for structured-data domains (e.g. graph-based instances), those properties inherent to the sort of data (e.g. common subgraphs, trees, cycles, etc.), might be important to achieve a competitive solution. This circumstance has motivated that some learning techniques which directly deal with structured data have been developed (multi-relational data mining [3]). Among them, distance-based methods are really popular for this purpose because several distance functions can be found for different sorts of data. However, unlike the ILP approaches, these methods do not give an explanation of their predictions. It is due to the fact that matches between two objects (e.g. two molecules) are encoded by a number (their distance). Unfortunately, model comprehensibility would be useful in some contexts. Imagine, for instance, in molecule classification, how interesting it would be to describe a cluster of molecules by saying what chemical structures these molecules have in common instead of saying that

* This work has been partially supported by the EU (FEDER) and the Spanish MEC under grant TIN 2004-7943-C04-02, ICT for EU-India Cross-Cultural Dissemination Project under grant ALA/95/23/2003/077-054, Generalitat Valenciana under grant GV06/301 and UPV under grant TAMAT.

they are closed according to a certain distance measure used in the clustering process.

Providing the possibility of this kind of descriptions for a distance-based algorithm would imply to incorporate a pattern language and generalisation operators [7]. But in this case, the model (expressed in a hopeful comprehensible pattern language) which generalises a set of elements could be inconsistent with the distance employed. The idea was initially considered in [4] by introducing the notion of distance-based binary generalisations. In [5], a more general framework in order to handle n-ary generalisation operators and to introduce the notion of minimal distance-based generalisations is presented.

In this paper, we use some of the concepts introduced in [5], to study generalisation operators and pattern languages for graph-based representations embedded in metric spaces. For this purpose, we consider two graph distance functions: the first one is described in [2, 8], and then we propose a second distance which is a bit more general. Next, a pattern language for graphs is introduced. This language is utilised to represent, in a comprehensible way, the generalisation computed by the generalisation operator. Next, for each metric space, a distance-based generalisation operator is defined according to our framework. Studying these generalisation operators will let us know how difficult extracting consistent patterns can be in each metric space.

2 Preliminaries

In this section we briefly review some basic concepts about graphs and graph distances. For any concept not explicitly included we refer the reader to [2].

A graph is a 4-tuple $G = (V, E, \mu, \nu)$ where V is a finite set of vertices, E is a set of edges (each one denoted as a pair of vertices belonging to $V \times V$), and μ and ν are functions which assign labels to vertices and edges respectively. The number of nodes of a graph $G = (V, E, \mu, \nu)$ is given by $|V|$ and it is denoted as $|V_G|$. The number of edges of a graph G is given by $|E|$ and is denoted as $|E_G|$. Given a graph $G = (V, R, \mu, \nu)$, a subgraph of G is a graph $S = (V_S, E_S, \mu_S, \nu_S)$ such that $V_S \subseteq V$, $E_S \subseteq E \cap (V_S \times V_S)$, and μ_S and ν_S are the restrictions of μ and ν to V_S and E_S respectively, that is $\mu_S(v) = \mu(v)$ (res. $\nu_S(e) = \nu(e)$) if $v \in V_S$ (res. $e \in E_S$) and undefined in otherwise.

A graph G_1 is isomorphic to a graph G_2 if there is an edge (and label) preserving bijection between all vertices in G_1 and G_2. Let G, G_1 and G_2 be graphs. G is a common subgraph of G_1 and G_2 if it is a subgraph of G_1 and G_2. A common subgraph G of G_1 and G_2 is maximal, denoted as $mcs(G_1, G_2)$, if there exists no other common subgraph G' such that G is a subgraph of G'. A subgraph G_1 of a graph $G = (V, E, \mu, \nu)$ is said to be induced by a set of vertices $W \subseteq V$ if for any pair of vertices w_1 and w_2 of W, (w_1, w_2) is an edge of G_1 if and only if $(w_1, w_2) \in E$, that is, G_1 is isomorphic to G. The concepts of common subgraph and maximal common subgraph are trivially extended to subgraphs induced by a set of vertices. We denote a maximal common subgraph of G_1 and G_2 induced by a set of vertices as $vimcs(G_1, G_2)$. That is, $vimcs$ looks

for a common set of vertices W in G_1 and G_2 such that W induces the maximal common subgraph and returns this common graph. Figure 1 illustrates with an example the above concepts. Note that, in general, there can be more than one maximal common subgraph induced by a set of vertices.

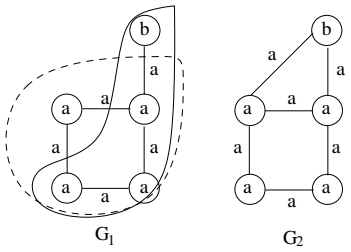


Fig. 1. G_1 is the maximal common subgraph of G_1 and G_2 whereas the subgraphs marked with dashed points and lines are the two maximal common subgraphs of G_1 and G_2 induced by a set of vertices.

We denote as d_1 and d_2 the distances we are working with. The first one is defined in terms of the minimal cost mapping transforming one graph into other. It is shown that d_1 can be expressed as $d_1(G_1, G_2) = |V_{G_1}| + |V_{G_2}| - 2|V_{vimcs(G_1,G_2)}|$ (see [2]). By modifying the definition of cost mapping employed in [2] we have derived a more general distance d_2 which can be calculated as $d_2(G_1, G_2) = |V_{G_1}| + |V_{G_2}| - 2|V_{mcs(G_1,G_2)}| + |E_{G_1}| + |E_{G_2}| - 2|E_{mcs(G_1,G_2)}|$.

Regarding the graphs G_1 and G_2 in Figure 1, we have $|V_{G_1}| = |V_{G_2}| = 5$, $|E_{G_1}| = 5$, $|E_{G_2}| = 6$, $V_{mcs(G_1,G_2)} = 5$, $E_{mcs(G_1,G_2)} = 5$ and $V_{vimcs(G_1,G_2)} = 4$. Hence, $d_1(G_1, G_2) = 2$ and $d_2(G_1, G_2) = 1$.

The computation of both distances in a feasible time might be a difficult task because the computation of *mcs* is needed and it is NP-complete. However, we are not interested in its calculus but we will use them to illustrate that extracting meaningful patterns in the metric space of graphs (G, d_2) is easier than in (G, d_1). The reason is that, as we will see, unlike *mcs*, the *vicms* of two graphs is not unique, and we would have to take all of them into account in order to define a distance-based generalisation operator in (G, d_1).

3 Distance-based generalisation operators

In this section we present the main concepts related to our proposal of a generalisation framework based on distances. For more details see [5].

Our approach aims to define generalisation operators for data embedded in a metric space (X, d). These operators are denoted as $\Delta(E)$, where E is a finite set of elements ($|E| > 2$) of X to be generalised. In principle, a generalisation of E will be a particular set in 2^X containing E. But, for the sake of comprehensibility, the generalisation computed by $\Delta(E)$ will be expressed by a pattern p belonging to a pattern language $\mathcal{L}$. In fact, every pattern p represents a set of elements of X and it is denoted by $Set(p)$. In this way, we can say that an element $x \in X$ is covered by a pattern p, if $x \in Set(p)$.

Additionally, if for every E, $\Delta(E)$ computes a generalisation of E "explaining" the distances among the elements of E, we will say that Δ is a distance-based generalisation operator. Then, the objective will be to find possible distance-based generalisation operators for the metric space (X, d).

For this purpose, we will focus on a particular kind of metric spaces, the connected ones. Informally speaking, this property means that given two elements in the metric space, we can always go from one to the other through elements belonging to the space such that these elements are at a minimal distance. In order to formally define what a connected space is, we need to introduce the notion of δ-path and the length of a δ-path.

Given a metric space (X, d), $I(X) = inf\{d(x, y) : \forall x, y \in X, x \neq y\}$ denotes the infimum distance of X. Let δ be a real positive number such that $\delta = I(X)$. Then, if $I(X) > 0$, a $\delta - path$ is a sequence of elements $P = \{x_i\}_{i=0}^{n>0}, \forall x_i \in X$, if $d(x_i, x_{i+1}) \leq \delta$ for all $0 \leq i \leq n-1$. Informally, if $(I(X) = 0)$, a $\delta - path$ is a continuous finite curve belonging to X. We also need the concept of the length of a δ-path P which is denoted by $L(P)$. If $P = \{x_i\}_{i=0}^{n>0}$, $L(P) = \sum_{i=0}^{n-1} d(x_i, x_{i+1})$. If P is a curve, $L(P)$ is just the length of the curve.

Let (X, d) be a metric space, two elements $x, y \in X$ are connected by a δ-path or equivalently, are δ-path connected, if there exists a δ-path $P = \{x_i\}_{i=0}^{n>0}$ such that $x_0 = x$ and $x_n = y$. Next, we will say that X is a connected metric space, if for every pair of elements x and y belonging to X, they are δ-path connected with $\delta \geq I(X)$. From this definition, we can say that a set $S \subseteq X$ is connected, if for every pair of elements in S they are δ-path connected, being $\delta = I(X)$. In what follows, we will use the term of x-connected space meaning that the metric space is connected and its $I(\cdot) = x$.

The notion of connected spaces and sets plays a key role in our approach since much too specific generalisations can be rejected (see Example 1).

Example 1. Let us suppose we are clustering graphs with the distance d_1 defined in Section 2. Imagine that each graph represents an organic compound and we would be interested in extracting some patterns saying which kind of molecules can be found in a cluster. One of the obtained clusters consists of two molecules m_1 and m_2 which are depicted in Figure 2.

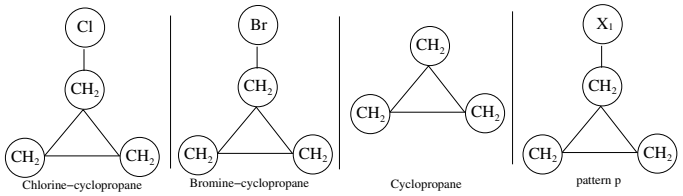


Fig. 2. The pattern p does not cover the cyclopropane molecule.

Let us obtain a pattern explaining the data distribution in this cluster. For this purpose, one could think of the pattern p (see Figure 2) saying "all the molecules with a cyclopropane structure and an extra atom". But this pattern might be much too specific. Considering that the molecules are really graphs

in the space (G, d_1), we could think that the pattern p overfits the data since the cyclopropane molecule, which would be placed "between" [1] m_1 and m_2, that is, $d_1(m_1, cyclopropane) = d_1(cyclopropane, m_2) = 1$, is not covered by this pattern. Perhaps, a more natural pattern would be that one saying "all the molecules built from cyclopropane".

The last reasoning can be modelled in terms of connections. We know that (G, d_1) is a 1-connected space [6], and clearly, the set given by $Set(p)$ is not connected because the elements m_1 and m_2 are not connected by means of a 1-path included in $Set(p)$. That is, m_1 and m_2 are, at least, 2-path connected since $d_1(m_1, m_2) = 2$. However, $Set($"all the molecules built from cyclopropane"$)$ would be connected.

Finally, in order to define distance-based generalisation operators, the concept of "nerve" of a set of elements E is needed. A nerve of E, denoted by $N(E)$, is simply a connected[2] graph whose nodes are the elements belonging to E. Now,

Definition 1. ***(Distance-based generalisation operator)*** *Let (X, d) be a connected metric space and let $\mathcal{L}$ be a pattern language. Given a mapping $\Delta : E \to p \in \mathcal{L}$, we will say that Δ is a (proper or hard) distance-based generalisation operator if, for every $E \subseteq X$, $E \subset Set(p)$, $Set(p)$ is a connected set and there exists a nerve $N(E)$ such that,*

- ***(proper)*** *For every pair of elements x, y in E such that they are directly linked in $N(E)$, $\Delta(E)$ includes* ***some*** *$I(X)$-path P connecting x and y such that $d(x, y) = L(P)$.*
- ***(hard)*** *For every pair of elements x, y in E such that they are directly linked in $N(E)$, $\Delta(E)$ includes* ***all*** *$I(X)$-path P connecting x and y such that $d(x, y) = L(P)$.*

Definition 1 can be difficult to understand. Let us see an example with a binary sets of elements $E = \{x, y\}$. In this case a proper distance-based operator will be that one computing a generalisation of E which includes some of the paths built from the elements placed "between" x and y. The generalisation is hard, if it includes all the paths built from the elements placed "between" of x and y. In what follows, we will refer to them as proper or hard operators. On the other hand, independently to the operator, we can say that a pattern generalises E in a proper or a hard way if $Set(p)$ satisfies the conditions above (for further details see [5]).

The distinction between proper and hard is due to the fact that hard generalisations explains the distance among the elements better than the proper ones because it takes into account all the elements "between" two given elements and not only some of them as proper generalisations do. In fact, in some cases, proper generalisations do not always have an intuitive interpretation in terms of the distance involved [5].

[1] Formally, given three elements x, y and z belonging to a metric space (X, d), we say that z is in "between" of x and y if $d(x, y) = d(x, z) + d(z, y)$.

[2] Here, the term connected refers to the well-known property for graphs.

4 Generalising set of graphs

In this section, we study some distance-based generalisation operators for graphs embedded in the metric spaces (G, d_1) and (G, d_2). First, for each metric space and using the pattern language $\mathcal{L}$ that will be defined below, we try to characterise the hard operators. From these hard operators, we will see how complicated the metric space is in order to compute distance-based patterns.

The pattern language $\mathcal{L}$ we are working with is composed of two types of patterns: the first-kind ($\mathcal{L}_1$) and the second-kind ($\mathcal{L}_2$) patterns. The so-called first-kind patterns will be a set of graphs built from an alphabet of labels containing constant and variable symbols. Regarding the second-kind patterns, these are expressed in terms of the first ones and they are specially introduced to improve the expressiveness of ($\mathcal{L}_1$).

Definition 2. ***(First-kind patterns*** $(\mathcal{L}_1)$***)*** *Given G the set of all the graphs over an alphabet of constant symbols A. If X is a set of variable symbols such that for all $X_i \in X$, X_i represents any constant symbol in A, then the language of first-kind patterns $(\mathcal{L}_1)$ is defined as the set of all the graphs over the alphabet $A \cup X$.*

Roughly speaking, the first-kind pattern is the intensional representation of a set of graphs sharing a particular topological structure, just as we show in the following example.

Example 2. Given the first-kind pattern language $(\mathcal{L}_1)$ defined from the set of constant symbols $A = \{a, b\}$ and the set of variable symbols $X = \{X_1, \ldots, X_n, \ldots\}$, consider the patterns p_1 in Figure 3.

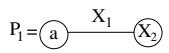


Fig. 3. An example of first-kind pattern.

This pattern represents only those graphs g in G made up of one edge and two vertices such that one of the vertices is labelled by the symbol a.

Although $\mathcal{L}_1$ permits quite interesting patterns, it still possesses some limitations. That is, imagine that we want to denote all the graphs in G having a subgraph in common. Despite the fact that this request seems usual, there is no pattern in $\mathcal{L}_1$ expressing it. For this reason, the class of the second-kind patterns is introduced.

Definition 3. ***(Second-kind patterns*** $\mathcal{L}_2$***)*** *Given the language of the the first-kind patterns $\mathcal{L}_1$, the language of the second-kind patterns $\mathcal{L}_2$ is defined as, $\mathcal{L}_2 = \{[p] : \forall p \in \mathcal{L}_1\} \cup \{\top\}$, where $[p]$ denotes all the graphs g in G having a subgraph covered by p and $\top$ denotes the whole space G.*

Example 3. The second-kind pattern p depicted in Figure 4 represents the set of all the graphs in G containing the path[3] $a - a - b$.

Next, let us define distance-based generalisation operators for (G, d_1) and (G, d_2).

[3] The concept of path referred here, is the well-known concept of path of a graph.

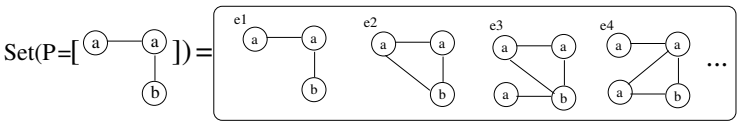


Fig. 4. An example of second-kind pattern using the notation $[\cdot]$.

4.1 Generalisation operators for (G, d_1)

Before defining a generalisation operator, we have proved that the metric space (G, d_1) is a 1-connected metric space [6]). Now, let us characterise hard generalisation operators in (G, d_1) using $\mathcal{L}$. Note that, for every finite set of graphs $\{g_i\}_{i=1}^{n>2}$ in G, an operator $\Delta(\{g_i\}_{i=1}^{n>2})$ will return a pattern belonging either to $\mathcal{L}_1$ or to $\mathcal{L}_2$. It can be shown that first-kind patterns do not represent connected sets (see [6]) and therefore, according to Definition 1 they cannot represent a set computed by a generalisation operator. Then, the only possibility is that Δ computes a second-kind pattern. Thus, a hard operator is given by (see [6]):

$$\Delta(\{g_i\}_{i=1}^{n>2}) = \begin{cases} [p] & \text{if conditions (1) and (2) hold} \\ \top & \text{otherwise.} \end{cases}$$

where conditions (1) and (2) are:

(1) p is a subgraph of the $mcs(\{r_i\}_{i=1}^{n})$, where $mcs(\{r_i\}_{i=1}^{n}) \neq \emptyset$ and each r_i denotes one of the possible $vimcs(\{g_i\}_{i=1}^{n})$.

(2) there exists a nerve $N(\{g_i\}_{i=1}^{n})$ such that for every pair of graphs, g_i and g_j, directly linked in the nerve N, all the possible $vimcs(g_i, g_j)$ are included in $\{r_i\}_{i=1}^{n}$.

As we can appreciate, the above conditions are extremely restrictive. For instance, regarding the two graphs depicted in the Figure 1, the pattern $[a-b]$ generalises G_1 and G_2 since they have the edge $a-b$ in common. However, this pattern is not hard because the common squared subgraph is an element "between" G_1 and G_2 but it is not covered by the pattern. In fact, p does not satisfy condition (1) since $[a-b]$ is not a common subgraph of the $vicms(G_1, G_2)$. This fact gives us an idea about how difficult is computing hard operators in this space. Imaging an algorithm implementing Δ, this would have to check if a subgraph of a set of graphs G is in its turn a subgraph of the $vicms(G)$. According to this observation, the algorithms in the literature approaching the maximum common subgraph among a set of graphs [1] cannot be used as an implementation of Δ because it can not be ensured that the returned subgraph belongs to the intersection of the $vicms$.

4.2 Generalisation operators for (G, d_2)

The metric space (G, d_2) is 1-connected as well [6]. One of the advantages of working with this metric space is that hard generalisation operators can be characterised in a more natural way using $\mathcal{L}$. Doing a similar analysis that in the previous subsection, hard operators can be defined in (G, d_2) as (see [6]):

$$\Delta(\{g_i\}_{i=1}^{n>2}) = \begin{cases} [p] & \text{if } p \text{ is a subgraph of the } mcs(\{g_i\}_{i=1}^{n\geq 2}) \\ \top & \text{otherwise.} \end{cases}$$

Unlike the space (G, d_1), the hard operators can be defined easier and implemented by using one of the several algorithms in the literature for searching common subgraphs among a set of graphs since now Δ directly uses the *mcs* of the set of graphs instead of the *vicms* of the set of graphs. Any subgraph returned by these algorithms can be perfectly used to define a hard generalisation.

5 Conclusions

Graph based learning is a challenging field due to the growing interest shown by several disciplines in mining data represented by means of graphs. However, most of the algorithms dealing with graphs, specially distance-based, do not return a model using graph features (e.x. common paths, walks, etc.) explaining why a sample has been labelled or grouped in one way. Although this kind of models is useful from a comprehensibility point of view, obtaining them could lead to inconsistency problems with the distance employed. In this work, we use some of the concepts of [5] to analyse which consistent models can be obtained when graphs are embedded in metric spaces.

References

1. E. Bengoetxea. Inexact graph matching using estimation of distribution algorithms, *PhD thesis*, 2003.
2. H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.
3. S. Dzeroski and N. Lavrac, editors. *Relational Data Mining*. 2001.
4. V. Estruch, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Distance based generalisation. In *Proc. of the ILP*, pages 87–102, 2005.
5. V. Estruch, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. On the relationship between distance and generalisation. Technical report, DSIC,UPV, 2006.
6. V. Estruch, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Some results about generalisations of graphs embedded in metric spaces. Technical report, DSIC,UPV, 2006.
7. A. Hotho and G. Stume. Conceptual clustering of text clusters. In *Proc. of the WS on Frachgruppentreffen Maschinelles Lernen, FGML*, pages 37–45, 2002.
8. A. Robles-Kelly and E.R. Hancock. Graph edit distance from spectral seriation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(3):365–378, 2005.

Conditional Random Fields for XML Trees

Florent Jousse[1], Rémi Gilleron[2], Isabelle Tellier[2], and Marc Tommasi[2]

INRIA Futurs and Lille University, LIFL, Mostrare Project
http://www.grappa.univ-lille3.fr/mostrare
[1]jousse@grappa.univ-lille3.fr, [2]first.last@univ-lille3.fr

Abstract. We present XML Conditional Random Fields (XCRFs), a framework for building conditional models to label XML data. XCRFs are Conditional Random Fields over unranked trees (where every node has an unbounded number of children). The maximal cliques of the graph are triangles consisting of a node and two adjacent children. We equip XCRFs with efficient dynamic programming algorithms for inference and parameter estimation. We experiment XCRFs on tree labeling tasks for structured information extraction and schema matching. Experimental results show that labeling with XCRFs is suitable for these problems.

1 Introduction

We address the task of labeling XML documents with Conditional Random Fields (CRFs). Many different problems in information science, such as information extraction, data integration, data matching and schema matching, are performed on XML documents and can be dealt with using XML labeling.

Lafferty *et al* have introduced CRFs in [4]. A CRF represents a conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$ with an associated graphical structure. CRFs have been successfully used in many sequence labeling tasks such as those arising in part-of-speech tagging [11], shallow parsing [10], named entity recognition [5] and information extraction [6, 8]; for an overview, see Sutton and McCallum's survey [9]. The idea of defining CRFs for tree structured data has shown up only recently. Basically, the propositions differ in the graphical structure associated with the CRFs. In [7], the output variables are independent. Other approaches such as [2, 12] define the graphical structure on rules of context-free or categorial grammars. Viola and Narasimhan in [14] consider discriminative context-free grammars, trying to combine the advantages of nongenerative approaches (such as CRFs) and the readability of generative ones. All these approaches apply to ranked rather than unranked trees. As far as we know, their graphical models are limited to edges.

We develop XCRFs, a new instance of CRFs that properly accounts for the inherent tree structure of XML documents. In an XML document, every node has an unlimited number of ordered children, and a possibly unbounded number of unordered attributes. The graphical structure for XCRFs is defined by: for ordered (parts of the) trees, the maximal cliques of the graph are all triangles

consisting of a node and two adjacent children; for unordered (parts of the) trees, the maximal cliques are edges consisting of a node and one child.

We define efficient dynamic programming algorithms for the inference problem and the parameter estimation problem in XCRFs. Because of the unranked property of XML trees, algorithms for XCRFs implement two recursions: an horizontal recursion following the child ordering and a vertical recursion following the sibling ordering.

We have implemented XCRFs in a system for labeling XML trees (`treecrf.gforge.inria.fr`). The system allows to label elements, attributes and text nodes of XML trees. In a first set of experiments, we show that attribute features and triangle features significantly improve the performance of XCRFs in XML tree labeling tasks. To the best of our knowledge, no alternative system for labeling trees exists, because so far only *ad hoc* solutions have been used. Nevertheless, in a second set of experiments, we have applied XCRFs on XML tree labeling tasks for schema matching and structured information extraction. For instance, for schema matching on the real estate domain, we show that the XCRF system performs really well although it does not use domain constraints or integrity constraints like the LSD system [3].

2 Conditional Random Fields

We refer to [9] for a complete introduction to CRFs. A CRF is a conditional distribution with an associated graphical structure. Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be two random fields, let $\mathcal{G}$ be an undirected graph over $\boldsymbol{Y}$. Let $\mathcal{C}$ be the set of all cliques of $\mathcal{G}$. The conditional probability distribution is of the form:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{c \in \mathcal{C}} \psi_c(\boldsymbol{y}_c, \boldsymbol{x})$$

where ψ_c is the potential function of the clique c and $Z(\boldsymbol{x})$ is a normalization factor. Each potential function has the form:

$$\psi_c(\boldsymbol{y}_c, \boldsymbol{x}) = \exp\Big(\sum_k \lambda_k f_k(\boldsymbol{y}_c, \boldsymbol{x}, c)\Big)$$

for some real-valued parameter vector $\Lambda = \{\lambda_k\}$, and for some set of real-valued feature functions $\{f_k\}$. This form ensures that the family of distributions parameterized by Λ is an exponential family. The feature function values only depend on $\boldsymbol{y}_c$, *i.e.* the assignments of the random variables in the clique c, and the whole observable $\boldsymbol{x}$. The two main problems that arise for CRFs are:

Inference: given an observable $\boldsymbol{x}$, find the most likely labeling $\hat{\boldsymbol{y}}$ for $\boldsymbol{x}$, *i.e.* compute $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x})$.

Training: given a sample set S of pairs $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}$, learn the best real-valued parameter vector Λ according to some criteria. In this paper, the criterion for training is the maximum conditional penalized log-likelihood.

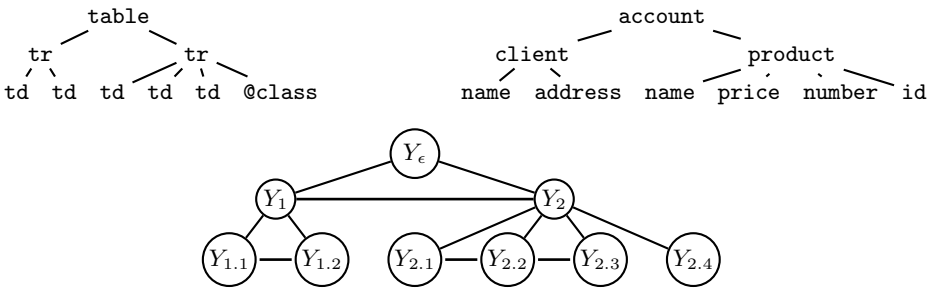


Fig. 1. An ordered unranked tree, its labeling and its graph.

3 CRFs for XML Trees

XML documents are represented by their DOM tree. We only consider element nodes, attribute nodes and text nodes of the DOM representation. Other types of nodes[1] are not concerned by labeling tasks. Attribute nodes are unordered, while element nodes and text nodes are ordered. We identify a node by a position which is a sequence of integers n and we denote by x_n the symbol in position n in the tree $\boldsymbol{x}$. The k ordered children of a node in position n are identified by positions $n.1$ to $n.k$. The unordered children are identified by positions $n.(k+1)$ and higher. As a running example, consider the two XML trees $\boldsymbol{x}$ (on the left) and $\boldsymbol{y}$ (on the right) in Figure 1. The set of nodes for both trees is $\{\epsilon, 1, 1.1, 1.2, 2, 2.1, 2.2, 2.3, 2.4\}$, ϵ being the root. The symbol in position 2.1 in $\boldsymbol{x}$ is `td`; in its labeling $\boldsymbol{y}$, the label in position 2.1 is `name`. $\{2.1, 2.2, 2.3, 2.4\}$ is the set of children of 2, where 2.1, 2.2 and 2.3 are ordered children and 2.4 is an unordered child of 2.

With every set of nodes, we associate a random field $\boldsymbol{X}$ of observable variables X_n and a random field $\boldsymbol{Y}$ of output variables Y_n where n is a position. The realizations of X_n will be the symbols of the input trees, and the realizations of Y_n will be the labels of their labelings. In the following, we freely identify realizations of these random fields with ordered unranked trees.

For their ordered parts, the structure of XML trees is governed by the sibling and the child orderings. We translate this structural property into XCRFs, by defining triangle feature functions that have the form:

$$f_k\big(y_n, y_{n.i}, y_{n.(i+1)}, \boldsymbol{x}, n.i\big) \ . \tag{3.1}$$

Their arguments are the labels assigned to the node n and to two consecutive children of it ($n.i$ and $n.(i+1)$), the whole observable $\boldsymbol{x}$, and the identifier of the clique in the tree $n.i$. For unordered parts of XML trees there is no next-sibling ordering, feature functions are thus only defined over nodes (node features) and pairs of nodes (edge features).

In Figure 1 (bottom), we show the graph for our running example. We denote by $\mathcal{C}$ the set of cliques in the dependency graph. We use f_k to index every

[1] comments, processing instructions...

feature function. To shorten the presentation, node, edge and triangle feature are all written like in (3.1). Each f_k is associated with a real-valued parameter λ_k, defining the vector $\Lambda = \{\lambda_k\}$. It is worth pointing out that the same set of feature functions with the same parameters is used for every clique in the graph. The conditional probability distribution for an XCRF can be written as:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{n.i \in \mathcal{C}} \exp\big(\sum_k \lambda_k f_k(y_n, y_{n.i}, y_{n.(i+1)}, \boldsymbol{x}, n.i)\big)$$

$$\text{where } Z(\boldsymbol{x}) = \sum_{\boldsymbol{y}} \Big(\prod_{n.i \in \mathcal{C}} \exp\big(\sum_k \lambda_k f_k(y_n, y_{n.i}, y_{n.(i+1)}, \boldsymbol{x}, n.i)\big)\Big)$$

Inference XCRFs are a particular case of graphical models. The treewidth of undirected graphs for XCRFs is 2. For every graph associated with an XCRF, a junction tree can be computed in linear time. Then the belief propagation algorithm can be applied [13, 11]. However using the knowledge of the tree-shaped graphical structures associated with XCRFs, we propose a dynamic programming algorithm for inference in XCRFs. The algorithm is based on the dynamic programming algorithm for probabilistic context-free grammars but introduces another set of variables due to the possibily large number of children.

Training Training an XCRF means learning its parameter vector Λ. We are given iid training data S of pairs of the form (observable tree, labeled tree). Parameter estimation is performed by penalized maximum likelihood. The conditional log-likelihood, defined as $\mathcal{L}_\Lambda = \sum_{(\boldsymbol{x},\boldsymbol{y})\in S} \log p(\boldsymbol{y}|\boldsymbol{x};\Lambda)$, is used. This function is concave and the global optimum is the vector of parameters with which the first derivative is null. However, finding analytically this derivative with respect to all the model parameters is impossible. The L-BFGS gradient ascent [1], which requires the computation of the partial derivatives of $\mathcal{L}_\Lambda$ for each parameter, is therefore used. To make these computations tractable, we introduce a dynamic programming algorithm using both forward-backward variables and inside-outside variables.

$Z(\boldsymbol{x})$ can be computed in $O(N \times M^3)$ where N is the number of nodes of $\boldsymbol{x}$ and M is the number of distinct labels in $\mathcal{Y}$. This result can be extended to the computation of the marginal probabilities in the gradient. This leads to an overall complexity for training in $O(N \times M^3 \times G)$ where N is the total number of nodes of the trees in the input sample S, M is the number of distinct labels in $\mathcal{Y}$, and G is the number of gradient steps. For linear chain CRFs only a factor M^2 occurs.

4 Experiments with the XCRF System

4.1 The XCRF System

The XCRF model is implemented by a freely available JAVA library[2]. For training, the parameters are estimated by maximizing the penalized log-likelihood. This implementation is a stochastic system which allows to label element, attribute and text nodes of XML trees. It provides the ability to not label some nodes by assigning them a label which means "not labeled". Labeling a medium-sized (about 1000 nodes) XML tree with an XCRF built on 100 features is almost immediate. An XCRF is specified by an XML file. Feature functions are 0-1 valued functions defined by XPATH expressions. There are node features, edge features, attribute features (edge features for unordered children) and triangle features.

4.2 Feature Generation

Users can easily introduce domain knowledge via the definition of feature functions in XCRFs. But to be fair, in all our experiments, feature functions are automatically generated from the training set, using a syntactic and domain-independent procedure. **Structure features** are node features, edge features and triangle features which are based on node symbols and labels. For instance, let 1_p be 1 if and only if p is true. Consider a tree $\boldsymbol{x} = \texttt{tr}(\texttt{td}, \texttt{td})$ and its labeling $\boldsymbol{y} = 0(1,2)$. The triangle feature $f(y_n, y_{n.i}, y_{n.(i+1)}, \boldsymbol{x}, n.i) = 1_{\{y_n=0\}}\ 1_{\{y_{n.i}=1\}}\ 1_{\{y_{n.(i+1)}=2\}}\ 1_{\{x_n=tr\}}\ 1_{\{x_{n.i}=td\}}\ 1_{\{x_{n.(i+1)}=td\}}$ is generated together with two edge features and three node features. **Attribute features** are based on attribute values. We also preprocess documents: additional information on the structure (number of children, depth, etc.) or on the textual content of a leaf (ContainsComma, ContainsColon, IsANumber, etc) are encoded into attributes. Attribute features are therefore generated from both original and preprocessed attributes. Consider a node n of x which is labeled with 0 and has an attribute a at position i whose value is 2 labeled with 1. An attribute feature $f(y_n, y_{n.i}, \boldsymbol{x}, n.i) = 1_{\{y_n=0\}}\ 1_{\{y_{n.i}=1\}}\ 1_{\{x_{n.i}=2\}}$ is generated.

4.3 Interest of Triangle Features

We first want to experimentally evaluate the significance of the triangle features, and to show that learning is successful with small datasets.

To evaluate our system we provide recall, precision and F1-measure over the number of nodes. Precision and recall are the ratio of the number of correctly labeled nodes to respectively the total number of labeled nodes and the total number of nodes which had to be labeled. Using a simple accuracy measure would take into account the nodes which are not labeled. The results would therefore be biased.

Triangle features. The "Courses" dataset[4] collected by Doan consists of 960 XML documents of about 20 nodes each, containing course information from five

[2] `http://treecrf.gforge.inria.fr/`

Features	F1	%docs
Edge	52.19	0
Edge & Attribute	84.91	26.51
Triangle	93.62	62.24
Triangle & Attribute	99.84	98.93

Nb Docs	F1	%docs
5	82.15	50.97
10	91.34	69.14
20	96.65	82
50	98.77	93.82

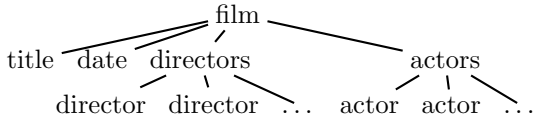
Table 1. Left: Triangle features versus edge features. **Right:** Varying the size of the training set

universities. All XML tags are removed and replaced with a unique and therefore uninformative one. The task consists in relabeling the XML documents with the original 14 distinct tags. We train XCRFs with node and edge structure features then with node, edge and triangle features. In both cases, we also train the XCRFs with or without attribute features. Results of Table 1 are means of 5 iterations in which XCRFs are trained on one fifth of the corpus and evaluated on the other 4 fifths. They show the F1-measure on document nodes and the percentage of XML documents that were completely correctly labeled. They confirm the relevance of triangle features. Indeed, results are far better with triangle features than when attribute features (containing also structural information) are added to edge features. Labeling is almost perfect with triangle and attribute features.

Number of examples. We consider the same task using structure features and attribute features. This time, we want to evaluate the impact of the number of examples in the learning set. Experimental results in Table 1 show that with only 20 documents for learning (192 in the previous experiment), the system already achieves more than 96% in F1-measure. When using more documents, the performances still rise, but very slowly. It is also interesting to note that when the XCRF has about 300 feature functions, the labeling of these documents is still immediate.

4.4 Experiments on Applicative Domains

Now, we apply our tree-labeling learning system to two applicative domains. The first one is structured information extraction, the second one is schema matching, both considered as a labeling task.



Structured Information Extraction. For this experiment, we take 558 XML documents from the MovieDB corpus used for the XML Document Mining Challenge[5]. The average number of nodes in these documents is 170. The task is to extract data according to the target DTD shown on the right. This purpose can be achieved by labeling the input XML documents according to this DTD. Note that two nodes with the same input tag might be labeled with different output tags, depending on the context. For instance, the tag `name` can be labeled either by `actor` or by `director`. To evaluate the XCRF

[4] `http://anhai.cs.uiuc.edu/archive`

Nb docs	Recall	Precision	F1	% docs
5	100	97.91	98.94	71.32
10	100	99.55	99.77	89.84
20	100	99.99	99.99	99.63

Table 2. Results on MovieDB Structured IE.

system on this task, we run 10 experiments, each time randomly choosing 5, 10 or 20 labeled documents on which an XCRF is trained and tested on the remaining documents. The results, very promising, are provided in Table 2.

Schema Matching. For the problem of schema matching, we evaluate XCRFs on the "Real Estate I" dataset[4], collected by Doan. This corpus, built from five real estate websites, describes house listing information and contains about 10000 documents of about 35 nodes each. Each of the five sources has its own schema. A unique mediated schema with 16 tags is also known. The task thus consists in labeling the nodes of the documents in their source schema with their corresponding tag in the mediated schema. As in [3], for every experiment, it is supposed that mappings are known for three sources out of the five ones and that documents labeled according to the mediated schema are the training set. We run 20 experiments, each time choosing at random 3 sources with which an XCRF with structure and attribute features is learnt. We took 5 labeled documents from each source. All the documents from the remaining 2 sources are used to evaluate the XCRF. The system achieved an excellent recall of 99%, and 88% of F1-measure. Unfortunately, these results can not be compared to the ones achieved by the LSD system given in [3]. Indeed, [3] measure the ratio of correct mappings between a tag in the source schema and its corresponding tag in the mediated schema. Since we are using a conditional model to label tree nodes, the same tag in the source schema can be mapped to different tags in the mediated schema depending on the context. Therefore, we can not provide this measure. However, it is still worth noting that we achieve very good results without using the domain knowledge, such as domain constraints or integrity constraints, that was used in the LSD system.

5 Conclusion

We have shown that XCRFs are a very relevant model for labeling XML trees. Experimental results show the significance of attribute and triangle features. Also, preliminary experimental results show that XCRFs perform very well on tasks such as structured information extraction or schema matching. Various extensions are being considered.

First, when labeling an XML tree, one can be interested in labeling inside the text nodes, *i.e.* assigning different labels to different parts of text. To do so, the use of both the tree structure and the text sequence is needed. Thus, combining

[5] `http://xmlmining.lip6.fr/Corpus`

linear chain CRFs and XCRFs could be a good way of taking advantage of both the structured and the linear view of XML documents.

Second, in the XCRF system, parameter estimation is done by maximizing the conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$, meaning we try to maximize the number of completely correctly labeled XML documents. Sometimes, for instance in schema matching, one might instead prefer to maximize the number of correctly labeled nodes. To do so, another criterion for learning could be the maximum pseudo-likelihood, which consists in maximizing the marginal probabilities $p(y_n|\boldsymbol{x})$.

References

1. Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Computing*, 16(5):1190–1208, 1995.
2. Stephen Clark and James R. Curran. Parsing the WSJ using CCG and log-linear models. In *Proc. of ACL*, pages 103–110, 2004.
3. AhHai Doan, Pedro Domingos, and Alon Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proc. of the ACM SIGMOD Conference*, pages 509–520, 2001.
4. John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*, pages 282–289, 2001.
5. A. McCallum and W. Li. Early results for named entity recognition with conditional random fields. In *Proc. of CoNLL'2003*, 2003.
6. David Pinto, Andrew McCallum, Xing Wei, and W. Bruce Croft. Table extraction using conditional random fields. In *Proc. of the ACM SIGIR*, pages 235–242, 2003.
7. S. Riezler, T. King, R. Kaplan, R. Crouch, J. Maxwell, and M. Johnson. Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques. In *Proc. of ACL*, pages 271–278, 2002.
8. Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In *Proc. of NIPS*, pages 1185–1192, 2004.
9. Charles Sutton and Andrew McCallum. *Introduction to Statistical Relational Learning*, chapter An Introduction to Conditional Random Fields for Relational Learning. MIT Press, lise getoor and ben taskar edition, 2006.
10. Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proc. of HLT-NAACL*, pages 213–220, 2003.
11. Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proc. of ICML*, pages 783–790, 2004.
12. Charles Sutton. Conditional probabilistic context-free grammars. Master's thesis, University of Massachusetts, 2004.
13. Ben Taskar, Pieter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *Proc. of UAI02*, pages 485–492, 2002.
14. Paul Viola and Mukund Narasimhan. Learning to extract information from semistructured text using a discriminative context free grammar. In *Proc. of the ACM SIGIR*, pages 330–337, 2005.
15. Hanna Wallach. Efficient training of conditional random fields. Master's thesis, University of Edinburgh, 2002.

Relational Sequence Alignment

Andreas Karwath and Kristian Kersting

University of Freiburg, Institute for Computer Science, Machine Learning Lab
Georges-Koehler-Allee, Building 079, 79110 Freiburg, Germany
{karwath,kersting}@informatik.uni-freiburg.de

Abstract. The need to measure sequence similarity arises in information extraction, music mining, biological sequence analysis, and other domains, and often coincides with sequence alignment: the more similar two sequences are, the better they can be aligned. Aligning sequences not only shows how similar sequences are, it also shows where there are differences and correspondences between the sequences.
Traditionally, the alignment has been considered for sequences of flat symbols only. Many real world sequences such as protein secondary structures, however, exhibit a rich internal structures. This is akin to the problem of dealing with structured examples studied in the field of inductive logic programming (ILP). In this paper, we propose to use well-established ILP distance measures within alignment methods. Although straight-forward, our initial experimental results show that this approach performs well in practice and is worth to be explored.

1 Introduction

Sequential data are ubiquitous and are of interest to many communities. Such data can be found in virtually all application areas of machine learning including computational biology, user modeling, speech recognition, empirical natural language processing, activity recognition, information extractions, etc. Therefore, it is not surprising that sequential data has been the subject of active research for decades. One of the many tasks investigated is that of *sequence alignment*. Informally speaking, a sequence alignment is a way of arranging sequences to emphasize their regions of similarity. Sequence alignments are employed in a variety of domains: in bioinformatics they are for instance used to identify similar DNA sequence, to produce phylogenetic trees, and to develop homology models of protein structures; in empirical language processing, they are for instance used for automatically summarizing, paraphrasing, and translating texts.

Most of the alignment approaches assume sequences of flat symbols. Many sequences occurring in real-world problems such as in computational biology, planning, and user modeling, however, exhibit internal structure. The elements of such sequences can be seen as atoms in a relational logic. The application of traditional alignment algorithms to such sequences requires one to either ignore the structure of the atoms, which results in a loss of information, or to take all possible combinations of arguments into account, which leads to a combinatorial explosion in the number of parameters.

The main contribution of the present paper is a general approach to align relational sequences, i.e., sequences of ground atoms. In particular, we propose to use well-established ILP distance measures within traditional alignment methods. Although straight-forward, our preliminary experimental results show that this approach performs well in practice and is worth to be explored.

We proceed as follows. After briefly reviewing alignment algorithms in Section 2, we discuss relational sequences and relational distance measures in Section 3. Before concluding, we present experimental results.

2 Sequence Alignment Algorithms

Alignment plays a major role in analyzing biological sequences. Consider e.g. the protein fold recognition problem, which is concerned with how proteins fold in nature, i.e., their three-dimensional structures. This is an important problem as the biological functions of proteins depend on the way they fold. Given a sequence of an unknown protein (query sequence) all approaches work in principle in a similar fashion: they scan an existing database of amino acids sequences (from more or less known proteins) and extract the most similar ones with regard to the query sequence. The result is usually a list, ordered by some score, with the best hits at the top of this list. The common approach for biologists, is to investigate these top scoring alignments or hits to conclude about the function, shape, or other features of query sequence.

One of the earliest alignment algorithm is that for global alignment by Needleman and Wunsch in 1970 [15]. The algorithm is based on dynamic programming, and finds the alignment of two sequences with the maximal overall similarity w.r.t. a given pairwise similarity model. In the biological domain, this similarity model is typically represented by pair-wise similarity or dissimilarity scores of pairs of amino acids. These scores are commonly specified by using a so-called similarity matrix, like the PAM [3] or BLOSUM [6] families of substitution matrices. The scores, or costs, associated with a match or mismatch between two amino acids, reflect to some extent the probability that this change in amino acids might have occurred over time of evolution.

More precisely, the Needleman-Wunsch algorithm proceeds as follows: initially, for two sequences of length m and n, a matrix with $m+1$ columns and $n+1$ rows is created. The matrix then is filled with the maximum score as follows:

$$M_{i,j} = \max \begin{cases} M_{i-1,j-1} + S_{i,j} & : \text{a match or mismatch} \\ M_{i,j-1} + w & : \text{a gap in the first sequence} \\ M_{i-1,j} + w & : \text{gap in the second sequence} \end{cases} \quad (1)$$

where $S_{i,j}$ is pairwise similarity of amino acids and w reflects a linear gap (insert step) penalty. The overall score of the alignment can be found in cell $M_{m,n}$.

To calculate the best *local* alignment of two sequences, one often employs the Smith-Waterman local alignment algorithm [19]. The main difference in this

algorithm when compared to the Needleman-Wunsch algorithm, is that all negative scores are set to 0. When visualizing the resulting alignment matrix, strands of non negative numbers correspond to a good local alignment. For both algorithms versions using affine gaps costs exist, i.e. one employs different kind of gap costs for opening a gap or for extending one. To discourage the splitting of connected regions due the enforcement of a gap in the middle of the alignment, commonly extra gaps are allowed to be inserted at the end and at the beginning at either no additional costs or relatively low costs (padding costs).

In general, the alignments resulting from an global or local alignment, show then the more *conserved* regions between two sequences. To enhance the detection of these conserved regions, commonly multiple sequence alignments are constructed. Given a number of sequences belonging to the same class, i.e. in biological terms believed to belong to the same family, fold, or are otherwise somehow related, alignments are constructed aligning all sequences in one single alignment, a so-called profile. A common approach for the construction of a multiple alignment is a three step approach: First, all pairwise alignments are constructed. Second, using this information as starting point a phylogenetic tree is created as *guiding tree*. Third, using this tree, sequences are joined consecutively into one single alignment according to their similarity. This approach is known as the neighbour joining approach [18].

A good overview of alignment algorithms, including construction of multiple alignments and the generation of phylogenetic trees, can be found in Durbin *et al.* [4].

3 Alignment of Sequences of Relational Objects

The alignment algorithms discussed in the previous section assume a given similarity measure $S_{i,j}$. Typically, this similarity measure is flat because the considered sequences consist of flat symbols. Many sequences occurring in real-world problems such as in computational biology, planning, and user modeling, however, exhibit internal structure. The elements of such sequences can elegantly be represented as objects in a relational logic (see e.g. [13] for an introduction to logic). For example, the secondary structure of the Ribosomal protein L4 can be represented as

$$\texttt{st}(\texttt{null},\texttt{short}),\texttt{he}(\texttt{h}(\texttt{right},\texttt{alpha}),\texttt{long}),\texttt{st}(\texttt{plus},\texttt{short}),\ldots,$$

representing helices of a certain type and length, `he`(*HelixType*,*Length*), and strands of a certain orientation and length, `st`(*Orientation*,*Length*). The symbols `st`, `null`, `short`, `he`, `h`, ... have an associated *arity*, i.e., number of arguments such as `st/2`, `he/2`, and `h/2` having arity 2, and `plus/0`, `1/0`, ... having arity 0. A *structured term* is a placeholder or a symbol followed by its arguments in brackets such as $\texttt{h}(\texttt{right},\texttt{X})$, `medium`, and $\texttt{he}(\texttt{h}(\texttt{right},\texttt{X}),\texttt{medium})$. A *ground term* is one that does not contain any variables such as $\texttt{st}(\texttt{null},\texttt{short})$, $\texttt{he}(\texttt{h}(\texttt{right},\texttt{alpha}),\texttt{long})$, ...

Relational sequence alignment simply denotes the alignment of sequences of such structured terms. More precisely, let $\mathtt{x} = \mathtt{x}_1, \ldots, \mathtt{x}_n$, $n > 0$, and $\mathtt{y} = \mathtt{y}_1, \ldots, \mathtt{y}_m$, $m > 0$, two sequences of logical objects and $d(i, j)$ a similarity measure indicating the score of aligning object $\mathtt{x}_i$ with object $\mathtt{y}_j$. Then, the *global alignment problem* seeks to find the match with highest score of both sequences in their entirety. The *local alignment problem* seeks to find the subsequence match with highest score.

Indeed, the only required task needed is to define the similarity measure $S_{i,j}$ in Equation (1). We propose to use one of the many distance measures developed within ILP [14]. As en example, consider one of the most basic measures proposed by Nienhuys-Cheng [16]. It treats ground structured terms as hierarchies, where the top structure is most important and the deeper, nested sub-structures are less important. Let $\mathcal{S}$ denote the set of all symbols, then Nienhuys-Cheng distance d is inductively defined as follows:

$$\begin{array}{ll} \forall \mathtt{c}/0 \in \mathcal{S} & d(\mathtt{c}, \mathtt{c}) = 1 \\ \forall \mathtt{p}/n, \mathtt{q}/m \in \mathcal{S} : \mathtt{p}/n \neq \mathtt{q}/m & d(\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_n), \mathtt{q}(\mathtt{s}_1, \ldots, \mathtt{s}_m)) = 0 \\ \forall \mathtt{p}/n \in \mathcal{S} & d(\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_n), \mathtt{p}(\mathtt{s}_1, \ldots, \mathtt{s}_n)) = \frac{1}{2\mathtt{n}} \sum_{\mathtt{i}=1}^{\mathtt{n}} d(\mathtt{t}_\mathtt{i}, \mathtt{s}_\mathtt{i}) \end{array}$$

To solve the corresponding relational alignment problem, we simply set $S_{i,j} = 1 - d(\mathtt{x}_\mathtt{i}, \mathtt{y}_\mathtt{i})$ in Equation (1). For sequences of more complex logical objects such as interpretations and queries, a different, appropriate similarity function has to be chosen. We refer to Jan Ramon's PhD Thesis [17] for a nice review of them.

4 Preliminary Experiments

Our intention here is to investigate to which extent relational sequence alignment is useful in real-world data sets. More precisely, we investigated the following two questions: **(Q1)** *Does the Nienhuys-Cheng measure provide better and more interesting alignments of sequences than a propositional one?* **(Q2)** *Is it possible to use relational sequence alignment for prediction purposes?* To this aim, we implemented the alignment method and the Nienhuys-Cheng distance measure in Python. In the following, we will describe some preliminary experiments carried out to investigate **Q1** and **Q2** and present their results.

4.1 Alignment of Protein Sequences

Here, we considered as real-world application the same data set as by Gutmann and Kersting [5], representing the five most populated folds in the SCOP class *Alpha and beta proteins (a/b)*. The examples are sequences of secondary structure elements of proteins which are similar in their three dimensional shape, but in general do not share a common ancestor (i.e. are not homologous). We have performed the experiments on the complete set of example proteins, as well as on a subset of proteins which do not share more than 40 per cent amino acid sequence identity (*cut 40*). This subset was generated using the ASTRAL database for the SCOP version 1.63 [1]. Overall, there are 2082 example sequences.

[1] http://astral.berkeley.edu/scopseq-1.63.html

Seq4	-	-	he_r_a_m	st_n_m	he_r_a_m	he_r_a_m	st_p_s	he_r_a_s	he_r_a_l	st_p_l
Seq3	he_r_a_l	he_r_3_s	he_r_a_l	st_n_s	he_r_a_s	he_r_a_m	st_p_s	he_r_a_l	st_p_s	he_r_a_s
Seq2	he_r_3_s	st_n_s	he_r_a_m	st_n_m	-	-	-	-	-	-
Seq1	st_n_m	st_p_m	he_r_a_l	-	st_p_m	he_r_a_m	st_p_s	he_r_a_m	st_p_s	he_r_a_s
(a)	*	*	*	*	*	*	*	*	*	*
(b)	*	*	**he(r,a,*)**	*	*	*	*	*	*	*

Seq4	-	-	he(r,a,m)	st(n,m)	he(r,a,m)	he(r,a,m)	st(p,s)	he(r,a,s)	-	-
Seq3	he(r,a,l)	he(r,3,s)	he(r,a,l)	st(n,s)	he(r,a,s)	he(r,a,m)	st(p,s)	he(r,a,l)	st(p,s)	he(r,a,s)
Seq2	he(r,3,s)	st(n,s)	he(r,a,m)	st(n,m)	he(r,3,s)	he(r,a,l)	st(p,m)	he(r,a,l)	-	-
Seq1	st(n,m)	st(p,m)	he(r,a,l)	st(p,m)	-	he(r,a,m)	st(p,s)	he(r,a,m)	st(p,s)	-
(c)	*	*	**he(r,a,*)**	**st(*,*)**	*	**he(r,a,*)**	**st(p,*)**	**he(r,a,*)**	*	*

Table 1. An alignment of four sequences using **(a)** the flat, **(b)** the back-translated flat, and **(c)** the relational approach. All symbols are abbreviated and the original alignment is truncated. The relational approach captures the conserved region much better than the flat ones as shown by the *lgg*-consensus sequences denoted in bold.

To answer question **Q1** we aligned sequences from one *fold* into a multiple alignment. Here we used the global alignment algorithm Needleman-Wunsch with affine gap penalties and seek to find *conserved* regions, i.e. subsequences in the multiple alignment, which express close similarity over all examples. The question of finding the appropriate gap costs in computational Biology is commonly answered by a trial and error approach. Here, we have solely concentrated on global alignments with affine gap costs using low padding costs. We have arbitrarily chosen the following gap costs: gap opening cost 1.5, gap extention cost 0.5, and padding cost 0.25.

To visualize the conserved regions, we have extracted the *consensus sequence* in form of the *lgg* (*least general generalization*) of all atoms in each particular position in the multiple alignment. An example of such a multiple alignment and the consensus sequence can be found in Table 4.1 **(c)**. Clearly, the consensus sequence reflects a conserved region of the four sequences.

The complete alignment possessed two conserved regions with a number of mismatches and gaps between. These conserved regions were not discovered when treating the sequence propositionally, i.e., each structured symbol $\texttt{st}(\texttt{null}, \texttt{short})$ is treated as a flat symbol `st_null_short`. In this case, using the Nienhuys-Cheng distance, only exact matches and pure mismatches are possible. Because none of the aligned flat symbols matches exactly, the resulting consensus sequence consist only of variables $*$, cf. Table 4.1 **(a)**. Even, when treating each of the aligned flat symbols as structured symbols again, the *lgg* consensus sequence does reveal much information, cf. Table 4.1 **(b)**. This affirmatively answers **Q1**.

The more informative consensus sequences, however, come at an expense: even apparent unrelated sequences get higher similarity scores. For instance, in our data set, we found sequences from different folds, where the relational alignment score is 4.75 times higher than the flat one. This could explain the

slightly lower predictive accuracy of the relational approach: a 10-fold cross-validated nearest neighbour classification (k=7) yielded 90.17% accuracy for the relational and 93.86% accuracy for the flat alignment approach for the complete dataset.

In any case, the predictive performances themselves are interesting. They are comparable to more sophisticated statistical relational learning results on similar data: LoHHMs 74.0% [8], Fisher kernels 84% [9], CRFs 92.96% [5]. This tends to affirmatively answer **Q2**.

For the *cut 40* subset, i.e. proteins in the five most populated classes not sharing more than 40 % amino acid sequence identity, the predictive performance decreases substantially for both representations: for the flat representation to 74.33 % and for the relational to 68.01 %. The reason for the decrease are obviously in the missing of close homologues in the cut 40 subset.

4.2 Alignment of Natural Language Sentences

Automatically paraphrasing sentences is of great practical importance for text-to-text NLP systems. Applications include text summarization and translation. For this task, Barzilay and Lee [1] proposed to use multiple (propositional) sequence alignment within clusters of similar sentences. Consider the following five sentences adapted from the example given by Barzilay and Lee:

1. **A** purple **latex balloon blew himself up in** ***a*** southern city ***Wednesday*****, bursting** two other balloons and deforming 27.
2. **A latex balloon blew himself up in** ***the*** area of Freiburg, on ***Sunday*****, bursting** itself and disfiguring seven balloons.
3. **A latex balloon blew himself up in** ***the*** coastal resort of ***Cuxhaven*****, bursting** three other balloons and deforming dozens more.
4. **A** purple **latex balloon blew himself up in** ***a*** garden cafe on ***Saturday*****, bursting** 10 balloons and deforming 54.
5. **A latex balloon blew himself up in** ***the*** centre of Berlin on ***Sunday*****, bursting** three balloons as well as itself and disfiguring 40.

The underlined sub-structures show the conserved regions computed by a propositional sequence alignment using the same gap costs as in the protein experiment; the bold parts denote the conserved regions of the relational sequence alignment; and italic parts denote the use of *lgg*s. The relational representation allows to encode additional information for each sentences. In particular, we used Brill's rule-based part of speech tagger, cf. [2], which is one of the most widely used tools for assigning parts of speech to words, to annotate each word with its part of speech tag. This yielded sequences such as

$$\texttt{dt(a)}, \texttt{jj(purple)}, \texttt{nn(latex)}, \texttt{nn(balloon)}, \texttt{vbd(blew)}, \texttt{prp(himself)}, \texttt{in(up)},$$
$$\texttt{in(in)}, \texttt{dt(a)}, \texttt{jj(southern)}, \texttt{nn(city)}, \texttt{nnp(wednesday)}, \texttt{comma}, \texttt{vbg(bursting)},$$
$$\texttt{cd(two)}, \texttt{jj(other)}, \texttt{nns(balloons)}, \texttt{cc(and)}, \texttt{vbg(deforming)}, \texttt{cd(27)}$$

Decreasing the gap opening costs to 0.5 resulted in

1. **A** purple **latex balloon blew himself up in** ***a*** southern city ***Wednesday*****, bursting** two other balloons **and** ***deforming*** 27.
2. **A latex balloon blew himself up in** ***the*** area of Freiburg, on ***Sunday*****, bursting** itself **and** ***disfiguring*** seven balloons.
3. **A latex balloon blew himself up in** ***the*** coastal resort of ***Cuxhaven*****, bursting** three other balloons **and** ***deforming*** dozens more.
4. **A** purple **latex balloon blew himself up in** ***a*** garden cafe on ***Saturday*****, bursting** 10 balloons **and** ***deforming*** 54.
5. **A latex balloon blew himself up in** ***the*** centre of Berlin on ***Sunday*****, bursting** three balloons as well as itself **and** ***disfiguring*** 40.

In both cases, the consensus regions of the propositional sequence alignments are proper sub-regions of the relational ones. This affirmatively answers **Q1**.

5 Related Work and Conclusions

Surprisingly few works investigated sequences of complex objects. Ketterlin [11] considered the clustering of sequences of complex objects but did not employ logical concepts. Likewise, Weskamp *et al.* [21] proposed an alignment algorithm for graphs. Lee and De Raedt [12] and Jacobs [7] introduced ILP frameworks for reasoning and learning with relational sequences. Recently, Tobudic and Widmer [20] used relational instance-based learning for mining music data, where sequential information is employed. To the best of our knowledge, however, the present paper proposes the first alignment approach for relational sequences, i.e., sequences of logical objects. The preliminary experimental results indicate that the relational sequences alignment reveals useful information in practice for different domains. That they are indeed more informative has been recently confirmed by Kersting and Karwath [10] using an information-theoretic, empirical argument on the protein data set.

The approach presented suggests a very interesting line of future research, namely to address the alignment of more complex logical objects such as interpretations, i.e., graphs. This has interesting applications e.g. in activity recognition, music mining, and plan recognition.

Acknowledgments: The authors thank Luc De Raedt for his support and Ross King for helpful discussions. The research was supported by the EU IST programme: FP6-508861, *Application of Probabilistic ILP II*; FP6-516169, *Inductive Queries for Mining Patterns and Models.*

References

1. R. Barzilay and L. Lee. Learning to Paraphrase: An Unsupervised Approach Using Multiple-Sequence Alignment. In *Proc. of HLT-NAACL-03*, pages 16–23, 2003.

2. E. Brill. Some advances in rule-based part of speech tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1994.
3. M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In M. O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, chapter 22, pages 345–352. Nat. Biomedical Research Foundation, 1978.
4. R. Durbin, S. Eddy, A. Krogh, and G. Mitchinson. *Biological Sequence Analysis*. Cambridge University Press, 1998.
5. B. Gutmann and K. Kersting. TildeCRF: Conditional Random Fields for Logical Sequence. In *Proceedings of the 15th European Conference on Machine Learning (ECML-06)*, 2006. (To appear).
6. S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl Acad. Sci.*, 89:10915–10919, 1992.
7. N. Jacobs. *Relational Sequence Learning and User Modelling*. PhD thesis, Computer Science Department, Katholieke Universiteit Leuven, Belgium, 2004.
8. K. Kersting, L. De Raedt, and T. Raiko. Logial Hidden Markov Models. *Journal of Artificial Intelligence Research (JAIR)*, 25:425–456, 2006.
9. K. Kersting and T. Gärtner. Fisher Kernels for Logical Sequences. In *Proc. of 15th European Conference on Machine Learning (ECML-04)*, pages 205 – 216, 2004.
10. K. Kersting and A. Karwath. On Relational Sequence Alignments and Their Information Content. Short paper to be presented at the 16th International Conference on Inductive Logic Programming (ILP06), 2006.
11. A. Ketterlin. Clustering Sequences of Complex Objects. In *Proc. of the 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD-97)*, pages 215–218, 1997.
12. S. D. Lee and L. De Raedt. Constraint Based Mining of First Order Sequences in SeqLog. In R. Meo, P. L. Lanzi, and M. Klemettine, editors, *Database Support for Data Mining Application*, pages 155–176. Springer, July 2004.
13. J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 2. edition, 1989.
14. S. H Muggleton and L. De Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*, 19(20):629–679, 1994.
15. S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–453, 1970.
16. S.-H. Nienhuys-Cheng. Distance between Herbrand interpretations: A measure for approximations to a target concept. In *Proc. of the 8. International Conference on Inductive Logic Programming (ILP-97)*, pages 250–260, 1997.
17. J. Ramon. *Clustering and instance based learning in first order logic*. PhD thesis, Department of Computer Science, K.U. Leuven, Leuven, Belgium, October 2002.
18. N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Evol. Biol*, 4(4):406–425, 1987.
19. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.
20. A. Tobudic and G. Widmer. Relational IBL in Classical Music. *Machine Learning*, 2006. (To be published).
21. N. Weskamp, E. Hllermeier, D. Kuhn, and G. Klebe. Graph Alignments: A New Concept to Detect Conserved Regions in Protein Active Sites. In R. Giegerich and J. Stoye, editors, *Proceedings German Conference on Bioinformatics*, pages 131–140, 2004.

Unbiased Conjugate Direction Boosting for Conditional Random Fields

Kristian Kersting and Bernd Gutmann

University of Freiburg, Institute for Computer Science, Machine Learning Lab,
Georges-Koehler-Allee, Building 079, 79110 Freiburg, Germany
{kersting,bgutmann}@informatik.uni-freiburg.de

Abstract. Conditional Random Fields (CRFs) currently receive a lot of attention for labeling sequences. To train CRFs, Dietterich *et al.* proposed a functional gradient optimization approach: the potential functions are represented as weighted sums of regression trees that are induced using Friedman's gradient tree boosting method. In this paper, we improve upon this approach in two ways. First, we identify an expectation selection bias implicitly imposed and compensate for it. Second, we employ a more sophisticated boosting algorithm based on conjugate gradients in function space. Initial experiments show performance gains over the basic functional gradient approach.

1 Introduction

Sequential data are ubiquitous and are of interest to many communities. Such data can be found in virtually all application areas of machine learning including computational biology, speech recognition, activity recognition, information extraction. Consider the protein secondary structure prediction problem [10], where the task is to assign a secondary structure class to each amino acid residue in the protein sequence. This is an instance of the general sequence labeling problem: assign labels $Y = \langle y_1 \dots y_n \rangle$ to a sequence $X = \langle x_1 \dots x_n \rangle$ of objects.

One appealing approach to label sequences are Lafferty *et al.*'s *conditional random fields* (CRFs) [6]. They are undirected models encoding the conditional dependency $P(Y|X)$ and have outperformed HMMs [11] on language processing tasks such as information extraction and shallow parsing. In contrast to generatively trained HMMs, the discriminatively trained CRFs are designed to handle non-independent input features such as such as the molecular weight and the neighboring acids of an amino acid.

The great flexibility to include a wide array of features raises one important questions: where do the features and the parameters come from? In many practical situations, even extremely simple undirected models such as the linear-chain CRFs considered in the present paper models can have severe training costs. Early approaches built large sets of feature conjunctions according to hand-built, general heuristics. This can result in extremely large feature sets, with millions of parameters trained on hundreds of thousands of training examples; parameter

estimation can literally take days [9]. Lafferty *et al.*'s [6] originally introduced iterative scaling algorithm for parameter estimation of a given CRF was reported to be exceedingly slow. Naive implementations of gradient ascent methods for maximizing the conditional likelihood $P(Y|X)$ have been proposed but they are typically quite slow too, because parameters highly interact. Therefore, it is not surprising that fast and integrated feature induction and parameter estimation techniques have been proposed.

McCallum's Mallet system [9] employs the BFGS algorithm, which is a second-order parameter optimization method that deals with parameter interactions, and induces features iteratively. Starting with a single feature, conjunctions of features are iteratively constructed that significantly increase conditional log-likelihood if added to the current model. Mallet's method is akin to the boosting idea [3] in that it creates new conjunctions (weak learners) based on a collection of misclassified instances, and assigns weights to the new conjunctions. Indeed, boosting has been applied to CRF-like models [1] and recently Dietterich *et al.* [2] presented a boosting approach, which is competitive to Mallet. It follows Friedman's gradient tree boosting algorithm [4], i.e., the potential functions are represented by sums of regression trees, which are grown stage-wise in the manner of Adaboost [3]. Each regression tree can be viewed as defining several new feature combinations, one corresponding to each path in the tree from the root to a leaf. Thus, the features can be quite complex; even relational conjunctions [5].

Despite elegancy, good performance, and flexibility, the gradient tree boosting approach, however, has two drawbacks. First, it imposes an expectation selection bias. In each boosting iteration, it generates functional gradient training examples for all possible label-label pairs at a sequence position. Thus, there are potentially quadratically many more expected than actually observed regression examples. We propose to quadratically raise the empirical frequency of observed label-label pairs. Second, only simple gradient ascent is employed so that maximization in one direction could spoil past maximizations. We propose the use of a more sophisticated boosting algorithm using conjugate directions. This way, we incorporate one of Mallet's major advantages into the functional gradient boosting approach: second-order information is used to adjust search directions so that previous maximizations are not spoiled. Experiments show for both modifications performance gains over the basic functional gradient approach.

We proceed as follows. After briefly reviewing CRFs and Dietterich *et al.*'s gradient tree boosting for training them, we show how to account for its expectation selection bias in Section 3. In Section 4, we devise the conjugate gradient variant. Before concluding, we experimentally evaluate our method in Section 5.

2 Linear-Chain CRFs

CRFs (see [6] for more details) are undirected graphical models that encode conditional probability distributions using a given set of features. In the present paper, we will focus on *linear-chain* CRF models.

Representation: Let G be an undirected graphical model over sets of random variables X and Y. For linear-chain CRFs, $X = \langle x_{i,j} \rangle_{j=1}^{T_i}$ and $Y = \langle Y_{i,j} \rangle_{j=1}^{T_i}$ correspond to the input and output sequences such that Y is a labeling of an observed sequence X. Now, they define the conditional probability of a state sequence given the observed sequence as $P(Y|X) = Z(X)^{-1} \exp \sum_{t=1}^{T} \Psi_t(y_t, X) + \Psi_{t-1,t}(y_{t-1}, y_t, X)$, where $\Psi_t(y_t, X)$ and $\Psi_{t-1,t}(y_{t-1}, y_t, X)$ are potential functions[1] and $Z(X)$ is a normalization factor over all state sequences X. Due to the global normalization by $Z(X)$, each potential has an influence on the overall probability.

Training: To apply CRFs, one most choose the representation for the potentials $\Psi_t(y_t, X)$ and $\Psi_{t-1,t}(y_{t-1}, y_t, X)$. Typically, it is assumed that the potentials factorize according to a set of features $\{f_k\}$, which are given and fixed, so that $\Psi(y_t, X) = \sum \alpha_k g_k(y_t, X)$ and $\Psi(y_{t-1}, y_t, X) = \sum \beta_k f_k(y_{t-1}, y_t, X)$ respectively. The model parameters are now a set of real-valued weights α_k, β_k; one weight for each feature. Furthermore, one must estimate the weights α_k, β_k. To do so, a conditional maximum likelihood approach is typically followed. That is, the (conditional) likelihood of the training data given the current parameter Θ_{m-1} is used to improve the parameters. Normally, one uses some sort of gradient search for doing this. The parameter in the next iteration are the current plus the gradient of the conditional likelihood function: $\Theta_m = \Theta_0 + \delta_1 + \ldots + \delta_m$ where $\delta_m = \eta_m - M \cdot \partial / \partial \Theta_{m-1} \sum_i \log P\left(y_i | x_i; \Theta_{m-1}\right)$ is the gradient multiplied by a constant η_m, which is obtained by doing a line search along the gradient.

Training via Gradient Tree Boosting: Dietterich *et al.*'s non-parametric approach interleaves both steps. More precisely, one starts with some initial potential Ψ_0, e.g. the zero function, and adds iteratively corrections $\Psi_m = \Psi_0 + \Delta_1 + \ldots + \Delta_m$. In contrast to the standard gradient approach, Δ_i denotes the so-called functional gradient, i.e.,

$$\Delta_m = \eta_m \cdot E_{x,y} \left[\partial / \partial \Psi_{m-1} \log P(y|x; \Psi_{m-1}) \right].$$

Since the joint distribution $P(x, y)$ is unknown, one cannot evaluate the expectation $E_{x,y}$. Dietterich *et al.* suggested to evaluate the gradient function at every position in every training example and fit a regression tree to these derived examples. More precisely, setting $F(y_{t-1}, y, t, X) = \Psi(y_t, X) + \Psi(y_{t-1}, y_t, X)$, the gradient becomes (see [2, 5] for more details),

$$\frac{\partial \log P(Y|X)}{\partial F(u, v, w_d(X))} = I(y_{d-1} \subseteq_\Theta u, y_d \subseteq_\Theta v) - P(y_{d-1} \subseteq_\Theta u, y_d \subseteq_\Theta v | w_d(X)),$$

where I is the indicator function, $\subseteq_\Theta$ denotes that u matches/subsumes y, and $P(y_{d-1} \subseteq_\Theta u, y_d \subseteq_\Theta v | w_d(X))$ is the probability that class labels u, v fit the class labels at positions $d, d-1$. By evaluating the gradient at every known position in the training data and fitting a regression model to these values, one gets an approximation of the expectation $E_{x,y} \left[\partial / \partial \Psi_{m-1} \right]$ of the gradient. In

[1] A *potential function* is a real-valued function that captures the degree to which the assignment y_t to the output variable fits the transition from y_{t-1} and X.

order to speed-up computations, not the complete input X is typically used but only a window $w_d(X) = x_{d-s}, \ldots, x_d, \ldots, x_{d+s}$, where s is a fixed window size.

In the following, we will present two improvements of the boosting approach.

3 Expectation Selection Bias

Reconsider the functional gradient. The expectation basically consist of two terms. The first term is the expected value of the potential function under the empirical distribution and the second term, which arises from the derivative of the normalization constant $Z(X)$, is the expectation of the potential function under the current model distribution. Due to the second term, the set S of generated examples the number K of given classes. For example, if we have 2 classes and 100 training sequences of length 200, then the number of training examples is $2^2 \times 200 = 80,000$. For 4 classes, there are already $4^2 \times 200 = 320,000$ examples. Most of these examples are likely to have never been observed. In the worst case, for each observed y_{t-1}, y_t pair, we additionally generate $K^2 - 1$ expected examples. Thus, the regression tree learner is biased towards reducing the variance of expected functional gradient training examples.

This suggests to quadratically raise the empirical frequency[2] of observed examples in S. To do so, we augment each functional gradient training example $(((w_t(X_i), k', \Delta(k, k', t))$ with an additional frequency weight f. We set f to $K^2 - 1$ if $y_{t-1} = k'$ and $y_t = k$, and to 1.0 otherwise. Furthermore, as we get more confident of the estimated model with increasing numbers of iterations t, we impose a decay of f using the inverted logistic function $(K^2 - 2) \cdot \exp[-c \cdot (t - 1)^b] + 1$, where c and b are constants; in our experiments $C = 0.007$ and $b = 4$. Within the regression tree learner, these weights are treated strictly as example multipliers. That is, the weighted variance is identical to the same analysis when the examples are replicated the specific number f of time.

4 Conjugate Direction Boosting

Reconsider the basic gradient-ascent optimization approach. One of the problems with choosing the step size η_m doing a line search is that a maximization in one direction could spoil past maximizations. This problem is solved by conjugate gradient boosting methods [7, 8]. Conjugate gradient boosting methods compute so-called conjugate directions $d_1, d_2, \ldots$ in the function space, which are orthogonal and hence do not spoil previous maximizations. The step size is estimated along these directions doing line searches.

More precisely, following Li *et al.*'s notation [7], conjugate gradient boosting iteratively performs two steps starting with setting the first direction to Δ_1:

[2] Weighting the empirical frequency has the appealing feature that it does not change the functional gradient values $\Delta(k, k', t)$. We only enforce a lower prediction variance over 'observed' training examples. This nicely fits our intuition that we are more confident in observed training examples in early iterations.

Algorithm 1 Conjugated Gradient Tree Boosting with line search.

1: **function** CGTreeBoost($Data, L$)
2: **for** $1 \leq m \leq M$ **do** ▷ Iterate Functional Gradient
3: $S := \emptyset$
4: **for** $1 \leq k \leq K$ **do** ▷ Iterate through the class labels
5: $S := S \cup$GenWeightedExamples$(k, Data, F_{m-1}, m)$ ▷ Generate examples
6: $\Delta_m :=$FitRelRegressTree(S, L) ▷ Functional gradient
7: **if** $m = 1$ **then**
8: $d_1 = \Delta_1$ ▷ Initial conjugate direction
9: **else**
10: $\beta_m = \frac{\langle \Delta_m, \Delta_m - \Delta_{m-1} \rangle}{\langle \Delta_{m-1}, \Delta_{m-1} \rangle}$ ▷ Polak-Ribiére formula
11: $d_m = \Delta_m + \beta_m \cdot d_{m-1}$ ▷ Next conjugate direction
12: $\eta_m :=$LineSearch$(Data, F_{m-1}, d_m)$ ▷ Line Search along d_m
13: $F_m := F_{m-1} + \eta_m \cdot d_m$ ▷ Model udpate
14: **return** F_M ▷ Return Potential

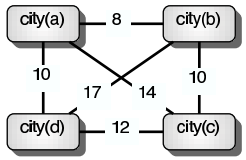
1. **(Conjugate directions)** Given the current gradient Δ_m, compute the empirical angle β_m between Δ_m and Δ_{m-1} on the training examples. The current gradient plus the old weighted gradient multiplied by the calculated angle is added to the current model, $d_m = \Delta_m + \beta_m \cdot d_{m-1}$. The angle β_m can be calculated by evaluating the Polak-Ribiére formula $\beta_m = \frac{\langle \Delta_m, \Delta_m - \Delta_{m-1} \rangle}{\langle \Delta_{m-1}, \Delta_{m-1} \rangle}$ for each example. Every weighted gradient d_m is a linear combination of the gradients $\Delta_1, \ldots, \Delta_m$. It can be shown that $d_t = \sum_{i=1}^{m} \beta_{i,m} \cdot \Delta_i$ where $\beta_{m,m} = 1$ and $\beta_{i,m} = \prod_{j=i+1}^{m} \beta_j$ if $i < m$.
2. **(Line search)** Compute the next model F_m by maximizing along the direction of d_m, i.e.

$$F_m = \sum_{k=1}^{m} \eta_k \cdot d_k = \sum_{i=1}^{m} \left(\sum_{j=1}^{m} \eta_j \cdot \beta_{i,j} \right) \Delta_i \ .$$

Training CRFs using conjugate gradient boosting is realized in CGTreeBoost in Alg. 1. Note that it uses the example weighting scheme discussed in the previous section. Furthermore, for the sake of simplicity and to stay close to traditional conjugate gradient for function optimization, we present the model update step as simply adding a single function. In fact, d_m is a linear combination of all previously computed functional gradients and one needs to keep track of the $\beta_{i,m}$. For a detailed discussion, we refer to [7, 8].

5 Preliminary Experiments

We implemented our approach in Yap 5.1.0 Prolog and investigate the following two questions: **Q1** *Can unbiased boosting speed-up the training of CRFs, i.e., faster improvements of the objective score, which is the conditional log-likelihood?* **Q2** *Can conjugate gradient boosting improve the training of CRFs?* To answer both questions, we carried out experiments on two domains, traveling salesman and protein secondary structure.



act	1	2	3	4	5	6	7	8
city(a)	-	7	2	1	10	-	-	2
city(b)	11	-	3	-	5	-	-	-
city(c)	12	8	-	-	5	8	10	3
city(d)	13	9	-	-	5	8	-	10

	Extra
city(a)	22
city(b)	50
city(c)	12
city(d)	10

Fig. 1. Traveling salesman instance used in the experiment.(Left) The map where nodes denote cities and edges transitions with associated costs. (Middle) Costs of activities. (Right) Costs of doing an activity fast in one city.

(Traveling Salesman [3]) There are n cities and actions $\mathcal{A}$, which can be done with normal speed or fast. All these actions have different costs. The task is, given is a sequence of activities $a_1, \ldots, a_T$, find a sequence of cities $c_1, \ldots, c_T$ such that the overall costs are minimized. We considered the instance with 4 cities and 8 actions as described in Figure 1. We randomly generated 100 independent activity sequences of length 15 and searched brute force for an optimal travel sequence. This yield relational sequences such as $X = \langle \texttt{act}(4, \texttt{normal}), \texttt{act}(1, \texttt{fast}), \texttt{act}(8, \texttt{normal}), \texttt{act}(7, \texttt{normal}), \ldots \rangle$ and $Y = \langle \texttt{city}(\texttt{a}), \texttt{city}(\texttt{d}), \texttt{city}(\texttt{c}), \texttt{city}(\texttt{c}), \ldots \rangle$. We refer to [5] for more details.

On a random 92/8 training/test split, we ran conjugate gradient tree (CGT) and gradient tree (GT) boosting with and without (,i.e., $\eta_m = 1.0$) line search on biased and unbiased examples. Figure 2 summarizes the results. As it can readily be seen, both algorithms overfit on this data set when doing a line search, cf. Fig 2 **(b)** and **(d)**. However, compensating for the expectation selection bias, cf. Fig 2 **(d)**, prevents CGT in contrast to GT from overfitting. Without line search, cf. Fig 2 **(a)** and **(c)**, CGT yields better performance than GT. This affirmatively answers **Q2**. More over, compensating for the expectation selection bias, cf. Fig. 2 **(c)**, results in larger improvements in early iterations of CGT and again prevents CGT from overfitting. This affirmatively answers **Q1**. The accuracies on the test set were in the same range across all algorithms.

(Protein Secondary Structure) This propositional data set was originally published by Qian and Sejnowski [10]. A protein consists of a sequence of amino acid residues. Each residue is represented by a single feature with 20 possible values (corresponding to the 20 standard amino acids). There are three classes: alpha helix, beta sheet, and coil (everything else). There is a training set of 111 sequences and a test set of 17 sequences. We used the same set up as in [5] and ran several variants of the basic algorithms.

The results summarized in Figure 3 **(a)** support the results of the first experiment: CGT performs better than GT and unbiased variants better than biased ones. Although doing a line search did not yield overfitting, unbiased CGT without line search still achieved the best predictive performance, partic-

[2] This domain has originally been used in [5] to show that CRFs for sequences of relational symbols can significantly outperform CRFs for sequences of flat symbols.

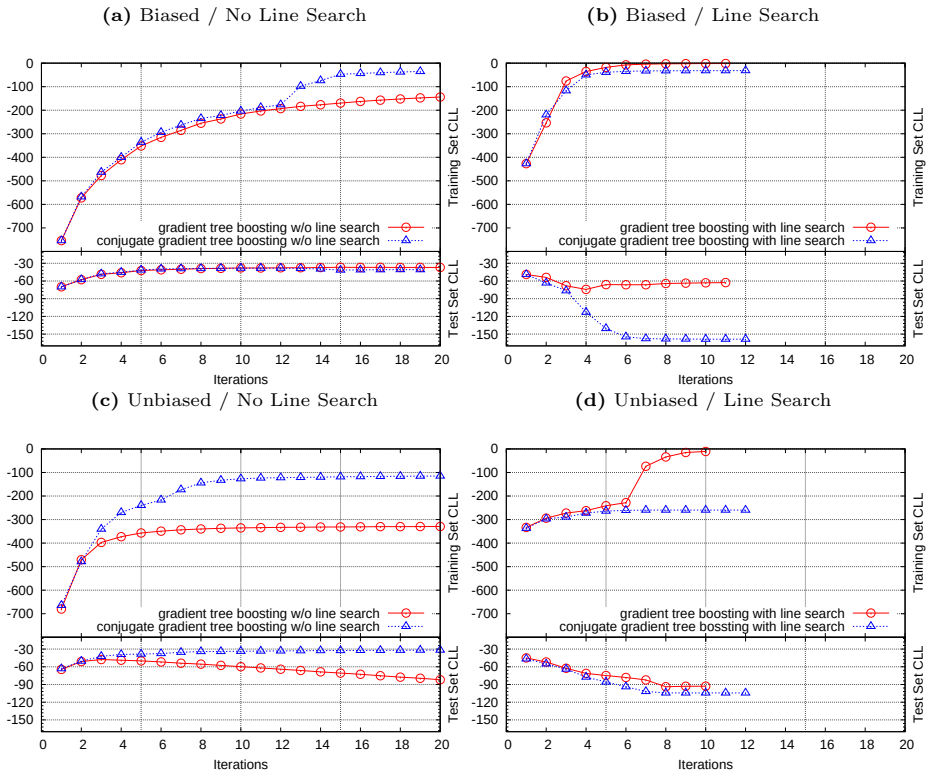


Fig. 2. Learning curves on the job scheduling domain. **(c, d)** compensated for the expectation selection bias in contrast to **(a, b)**. Cases **(b, d)** used a line search whereas **(a,c)** did not

ularly compared to the original GT, cf. Figure 3 **(b)**. This affirmatively answers **Q1** and **Q2**.

6 Conclusions

Training CRFs can be viewed as gradient ascent in function space. In this paper, we devised a novel algorithm for training CRFs, which employs conjugate directions in function space. Furthermore, we identified an expectation selection bias when training CRFs and presented an example weighting approach to compensate for it. Preliminary experiments are encouraging: the resulting approach can indeed perform better than simple gradient tree boosting. To validate this, further experiments should be conducted.

Acknowledgments The authors thank Luc De Raedt for his support, Alan Fern for interesting discussions, and Tom Dietterich for providing his version of

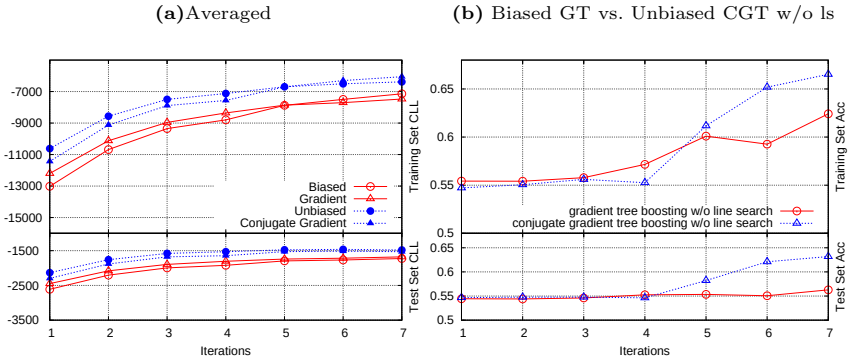


Fig. 3. Learning curves on the protein secondary structure domain. **(a)** Average over several variants of biased/unbiased and GT/CGT. **(b)** Predictive accuracies of unbiased CGT and biased GT both w/o line search.

Qian and Sejnowski's benchmark. The research was supported by the European Union IST programme: FP6-508861, *Application of Probabilistic ILP II*.

References

1. Y. Altun, T. Hofmann, and M. Johnson. Discriminative learning for label sequences via boosting. In *Advances in Neural Inf. Proc. Systems (NIPS-15)*, 2003.
2. T. Dietterich, A. Ashenfelter, and Y. Bulatov. Training conditional random fields via gradient tree boosting. In *Proc. 21st International Conf. on Machine Learning*, pages 217–224. ACM, 2004.
3. Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *In Proceedings of ICML-96*, pages 148–156. Morgan Kaufman, 1996.
4. J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29, 2001.
5. B. Gutmann and K. Kersting. TildeCRF: Conditional Random Fields for Logical Sequences. In *Proc. of the 17th European Conference on Machine Learning (ECML-06)*, 2006. (To appear).
6. J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th Int. Conf. on Machine Learning (ICML-01)*, pages 282–289, 2001.
7. L. Li, Y. S. Abu-Mostafa, and A. Pratap. CGBoost: Conjugate Gradient in Function Space. Technical Report CaltechCSTR:2003.007, Calfornia Institute of Technology, August 2003.
8. R. W. Lutz and P. Bühlmann. Conjugate direction boosting. *Journal of Computational and Graphical Statistics*, 15(2):287–311, 2006.
9. A. McCallum. Effciently inducing features of conditional random fields. In *Proc. of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-03)*, 2003.
10. N. Quian and T. J. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *JMB*, 202:865–884, 1988.
11. L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–285, 1989.

Tree Kernel Engineering for Proposition Re-ranking

Alessandro Moschitti, Daniele Pighin, and Roberto Basili

Department of Computer Science
University of Rome "Tor Vergata", Italy
{moschitti,basili}@info.uniroma2.it
daniele.pighin@gmail.com

Abstract. Recent work on the design of automatic systems for semantic role labeling has shown that such task is complex from both modeling and implementation point of views. Tree kernels alleviate such complexity as kernel functions generate features automatically and require less software development for data pre-processing. In this paper, we study several tree kernel approaches for boundary detection, argument classification and, most notably, proposition re-ranking. The comparative experiments on Support Vector Machines with such kernels on the CoNLL 2005 dataset show that very simple tree manipulations trigger automatic feature engineering that highly improves accuracy and efficiency in every SRL phase.

1 Introduction

A lot of attention has been recently devoted to the design of systems for the automatic labeling of semantic roles (SRL) as defined in two important projects: FrameNet [1], inspired by Frame Semantics, and PropBank [2] based on Levin's verb classes. SRL is a complex task consisting in the recognition of predicate argument structures within natural language sentences.

Research on the design of automatic SRL systems has shown that (shallow or deep) syntactic information is necessary to achieve a good accuracy, e.g. [3,4]. A careful analysis of literature features encoding such information reveals that most of them are fragments of syntactic trees of training sentences. Thus, a natural way to represent them is the adoption of tree kernels as described in [5]. Tree kernels show important advantages: first, we can implement them very quickly as the feature extractor module only requires the writing of the procedure for subtree extraction. In contrast, traditional SRL systems are based on the extraction of more than thirty features [6], which require the writing of at least thirty different procedures. Second, combining tree kernels with a traditional attribute-value SRL system allows us to obtain a more accurate system. Usually the combination of two traditional systems (based on the same machine learning model) does not result in an improvement as their features are more or less equivalent as shown in [4]. Finally, the study of the effective structural features

can inspire the design of novel linear features, which can be used with a more efficient model (i. e. linear SVMs).

In this paper, we carry out tree kernel engineering [7,8] to increase the accuracy of the boundary detection, argument classification and proposition re-ranking steps. In the first two cases (Section 2.1), the engineering approach relates to marking the nodes of the encoding subtrees to generate substructures more strictly correlated with a particular argument, boundary or predicate. For the latter case (Section 2.2), i. e. proposition re-ranking, we try both marking large parts of the tree that dominates the whole predicate argument structure and utterly reworking the syntactic structure. Our extensive experimentation of the proposed tree kernels with Support Vector Machines on the CoNLL 2005 data set provides interesting insights on the design of performant SRL systems (Section 3).

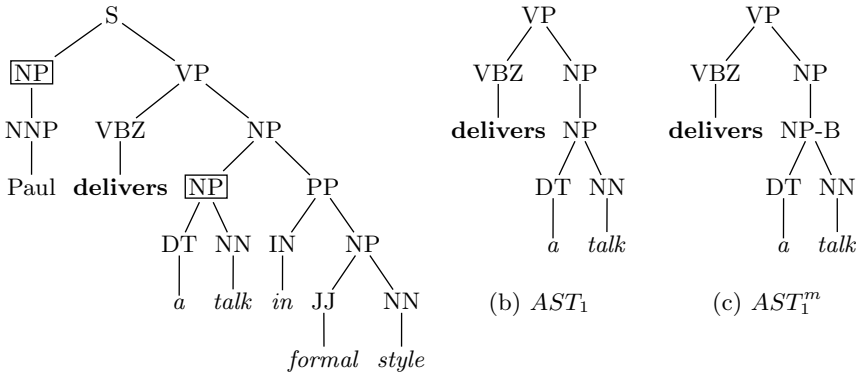
2 A Model for Semantic Role Labeling

The SRL approach that we adopt is based on the deep syntactic parse [9] of the sentence that we intend to semantically annotate. The standard algorithm concerns the classification of tree node pairs $\langle p, a\rangle$, where p is the node that exactly dominates the target predicate and a is the node dominating a potential argument. If $\langle p, a\rangle$ is selected as an argument, then the leaves of the tree rooted in a will be considered as the words constituting such argument. There are hundreds of pairs in a sentence, thus, if we use training corpora containing hundreds of thousands of sentences, we have to deal with millions of instances.

To limit such complexity, we can divide the problem in two subtasks: (a) boundary detection, in which a single classifier is trained on many instances to detect if a node is an argument or not, i. e. if the sequence of words dominated by the target node constitutes a correct boundary; and (b) argument classification, in which only the set of nodes corresponding to correct boundaries are considered. These can be used to train a multiclassifier that, for such nodes, selects the most appropriate labeling. For example, n classifiers can be combined with a One-vs-All approach, selecting for each argument node the role associated with the maximum among the n scores provided by the individual role classifiers.

The main advantage of this approach is the use of just one computationally expensive classifier, i. e. the one for boundary detection. Regarding the feature representation of $\langle p, a\rangle$, we can extract syntactic fragments from the sentence-parse tree proposed in [3], e.g. the *Phrase Type* or *Predicate Word*. An alternative to the manual fragment extraction is the use of Tree Kernels as suggested in [5]. Tree kernels are especially useful when the manual design of features is made complex by the use of a re-ranking module. This has been shown to be essential to obtain state-of-the-art performance [10].

The next sections describe our tree kernel approaches for the classification of boundaries and arguments and the re-ranking of complete predicative structures.



(a) A parse tree (b) AST_1 (c) AST_1^m

Fig. 1. Syntactic parse tree of the sentence *John delivers a talk in formal style* (a), AST_1 (b) and AST_1^m (c) for the argument A1 *a talk*.

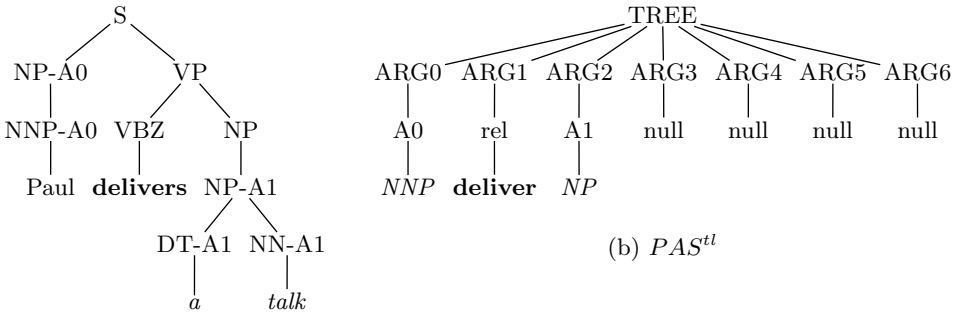
2.1 Kernels for Boundary Detection and Argument Classification

Once a basic kernel function is defined, we need to characterize the predicate-argument pair with a subtree. This allows the function to generate a large number of syntactic features related to such pair. The approach proposed in [5] selects the minimal subtree that includes a predicate with one of its arguments. For example, Figure 1(a) shows the parse tree of the sentence *Paul delivers a talk in formal style* whereas Frame (b) illustrates the AST_1 subtree that characterizes the predicate *to deliver* with its argument A1 *a talk*.

AST_1s are very effective for argument classification but not for boundary detection since two nodes that encode correct and incorrect boundaries may generate very similar AST_1s [5]. To solve this problem, we simply mark the argument node with the label *B*, denoting the boundary property. This new subtree is called a marked argument spanning tree (AST_1^m) and it is shown in Figure 1(c). A positive example for the AST_1^m classifier is a subtree in which the marked node exactly covers the boundaries of an argument, whereas the marking of any other node within the same subtree results in a negative example.

2.2 Tree Kernels for the Proposition Re-ranking Task

Our re-ranking mechanism is similar to that described in [11], where a Viterbi algorithm is used to evaluate the most likely labeling schemes for a given predicate and a re-ranking mechanism selects the best annotation. The re-ranker is a binary classifier trained with pairs $\langle s_i, s_j \rangle$ where s_i and s_j are taken from the set of the most m probable prepositions output by the Viterbi algorithm for the same target predicate. The classifier is meant to output a positive value if s_i is more



(a) AST_n^{cm} (b) PAS^{tl}

Fig. 2. AST_n^{cm} and PAS^{tl} representations of the example proposition.

accurate that s_j and a negative value otherwise. Each candidate proposition s_i can be described by a structural feature t_i and by a vector of linear features v_i representing information that cannot be captured by t_i, e.g. the probability associated with the annotation output by the Viterbi algorithm. As a whole, each classifier example e_i is described by a tuple $\langle t_i^1, t_i^2, v_i^1, v_i^2 \rangle$, where $\langle t_i^1, v_i^1 \rangle$ and $\langle t_i^2, v_i^2 \rangle$ describe the first and second candidate annotations, respectively.

Using the above tuple, we can define the following kernels:

$$K_{tr}(e_1, e_2) = K_t(t_1^1, t_2^1) + K_t(t_1^2, t_2^2) - K_t(t_1^1, t_2^2) - K_t(t_1^2, t_2^1)$$
$$K_{pr}(e_1, e_2) = K_p(v_1^1, v_2^1) + K_p(v_1^2, v_2^2) - K_p(v_1^1, v_2^2) - K_p(v_1^2, v_2^1)$$

where K_t is a tree kernel function defined in [12] and K_p is a polynomial kernel applied to the feature vectors. The final kernel that we use for re-ranking is the following:

$$K(e_1, e_2) = \frac{K_{tr}(e_1, e_2)}{|K_{tr}(e_1, e_2)|} + \frac{K_{pr}(e_1, e_2)}{|K_{pr}(e_1, e_2)|} .$$

Among the many different structural features that we tested with our re-ranker, the most effective are the completely marked argument structure spanning tree (AST_n^{cm}) and the lemmatized type-only predicate argument structure (PAS^t).

An AST_n^{cm} (see Figure 2(a)) consists of the node spanning tree embracing the whole argument structure: each argument node's label is enriched with the role assigned to the node by the role multiclassifier, the labels of the descendants of each argument node being accordingly modified down to pre-terminal nodes. Marking the nodes' descendants is meant to force substructures to match only among homogeneous argument types. This representation is meant to provide rich syntactic and lexical information about the parse tree encoding the predicate structure.

A PAS^{tl} (see Figure 2(b)) is a completely different structure that represents the syntax of the predicate argument structure, i. e. the number, type and position of each argument, minimizing the amount of lexical and syntactic information derived from the parse tree. The syntactic links between the argument nodes are represented as a fake 1-level tree, which is shared by any PAS^{tl} and therefore does not influence the evaluation of similarity between pairs of structures. Such structure accommodates sequentially all the arguments of an annotation, each slot being attached a pre-terminal node standing for the node type and a terminal symbolizing the syntactic type of the argument node. In general, a proposition consists of m arguments, with $m < 7$. In this case, all the nodes ARGi, $i \leq m \leq 6$ are attached a dummy descendant marked *null*. The predicate is represented by means of a pre-terminal node labeled *rel* to which the lemmatization of the predicate word is attached as a leaf node.

Table 1. Correct (+), incorrect (-) and overall (tot) number of potential argument nodes from sections 2, 3 and 24 of the PropBank.

Section 2			Section 3			Section 24		
+	-	tot	+	-	tot	+	-	tot
12,741	185,178	197,919	7,023	139,823	146,846	8,234	130,489	138,723

Table 2. Performance improvement on the boundary detection and argument classification tasks using engineered tree kernels.

	Boundary detection	Argument classification
AST_1	75.24	82.07
AST_1^m	75.06	77.17

3 Experiments

In these experiments we evaluate the impact of our proposed kernels on the different phases of the SRL task. The resulting accuracy improvement confirms that the node marking approach enables the automatic engineering of effective SRL features.

The empirical evaluations were carried out within the setting of the CoNLL-2005 Shared Task [4] described in `www.lsi.upc.edu/∼srlconll/` by means of SVM-light-TK available at `http://ai-nlp.info.uniroma2.it/moschitti/` which encodes fast tree kernel evaluation [12] in SVM-light [13]. We used a regularization parameter (option `-c`) equal to 1 and $\lambda = 0.4$ (see [5]).

3.1 Boundary Detection and Argument Classification Results

For the boundary detection experiments we used Section 02 for training and Section 24 for testing, whereas for argument classification also Section 03 was

used for training. Their characteristics in terms of potential argument nodes[1] are shown in Table 1.

Table 3. Number of distinct annotations output by the Viterbi algorithm and of pair comparisons (i. e. re-ranker input examples) in the PropBank sections used for the experiments.

	Section 12	Section 23	Section 24
Annotations	24,494	26,325	16,240
Comparisons	74,650	81,162	48,582

Table 4. Summary of the proposition re-ranking experiments with different training sets.

Training section	AST_n^{cm}	PAS^{tl}	PAS^{tl} + STD
12	-	78.27	77.61
24	76.47	78.15	77.77
12+24	-	78.44	-

The results obtained using the AST_1 and the AST_1^m based kernels are reported in Table 2 in rows 2 and 3, respectively. Columns 2 and 3 show their respective performance (in terms of F_1 measure) on the boundary detection and argument classification phases. We note that: (1) on boundary detection, AST_1^ms improve the F_1 over AST_1 by about 7 points, i. e. 82.07 vs. 75.24. This suggests that marking the argument node simplifies the generalization process; (2) using an engineered tree kernel also improves the argument classification task by about 2 points, i. e. 77.17 vs. 75.06. This confirms the outcome on boundary detection experiments and the fact that we need to distinguish the target node from the others.

3.2 Proposition Re-ranking Results

For our proposition re-ranking experiments, Section 23 was used for testing. On such test set, considering the first 5 alternatives output by the Viterbi algorithm, our model has a lower bound of the F_1 measure of 75.91 (corresponding to the selection of the first alternative, i. e. the most likely with respect to the probabilistic model) and an upper bound of 84.76 (corresponding to the informed selection of the best among the 5 alternatives, i. e. the theoretical output of a perfect re-ranker). The number of distinct annotations output by the Viterbi

[1] As the automatic parse trees contain errors, some arguments cannot be associated with any covering node. This prevents us to extract a tree representation for them. Consequently, we do not consider them in our evaluation. In sections 2, 3 and 24 there are 454, 347 and 731 such cases, respectively.

algorithm for each section that we used is shown in Table 3, Row 2. In Row 3, the number of pair comparisons, i. e. the number of training/test examples for the classifier.

Table 4 summarizes the outcome of our experiments. First, we compared the accuracy of the AST_n^{cm} and PAS^{tl} classifiers trained on Section 24 (in Row 3, Columns 2 and 3) and discovered that the latter structure produces a noticeable F_1 improvement, i. e. 78.15 vs. 76.47. Second, we added the local (to each argument node) linear features commonly employed for the boundary detection and argument classification tasks, as in [10] to the PAS^{tl} kernel (Column 4). The comparison with the simple PAS^{tl} on 2 different training sets (Rows 2 and 3) shows that the introduction of the standard linear features produces a performance decrease on both sections 12 and 24 . Finally, we trained our best re-ranking kernel, i. e. the PAS^{tl}, with both sections 12 and 24 achieving an F_1 measure of 78.44 (Row 4).

These results suggest that: (1) the re-ranking task is very difficult from a ML point of view: in fact, adding or removing thousands of training examples has only a small impact on the classification accuracy; (2) the PAS^{tl} kernel is much more effective than the AST_n^{cm} one, which is always outperformed. This may be due to the fact that two AST_n^{cm}s always share a great number of substructures, since most alternative annotations tend to be very alike and the small differences among them only affect a small part of their enriched syntactic parse trees; (3) on the other hand, the little amount of local parsing information encoded in the PAS^{tl}s allows for a good generalization process; (4) the introduction of the standard, local linear features in our re-ranking model caused a performance loss of about 0.5 points on both Sections 12 and 24. This fact, which is in contrast with what has been shown in [10], might be the consequence of the small training sets that we employed. In fact, local linear features tend to be very sparse and their effectiveness should be evaluated against a larger data set.

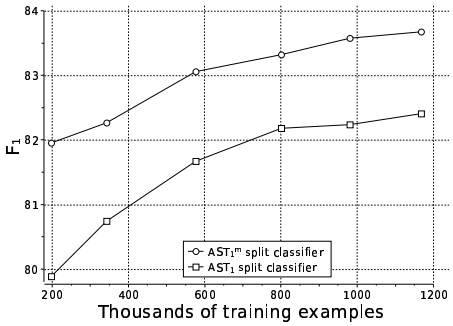


Fig. 3. Learning curve comparison for the boundary detection phase between the AST_1 and AST_1^m F_1 measures.

4 Conclusions

The design of automatic systems for the labeling of semantic roles requires the solution of complex problems. Among others, feature engineering is made difficult by the structural nature of the data, i. e. features should represent information contained in automatic parse trees. A system based on tree kernels alleviate such complexity as kernel functions can automatically generate effective features.

In this paper, we have improved tree kernels by studying different strategies, e.g. AST_1^ms highly improve accuracy in both the boundary detection (about 7%) and argument classification subtasks (about 2%). We have also engineered different structured features for the re-ranking module, which improves our system of about 2.5 percent points. This is quite a good results as it approaches the state-of-the-art using only a small fraction of all the available data. In the near future, we would like to use more such data along with other kernels described in [12].

Acknowledgments

This research is partially supported by the European project, PrestoSpace (FP6-IST-507336).

References

1. Johnson, C.R., Fillmore, C.J.: The framenet tagset for frame-semantic and syntactic coding of predicate-argument structure. In proceedings of NAACL 2000, Seattle WA. (2000)
2. Kingsbury, P., Palmer, M.: From Treebank to PropBank. In proceedings of LREC'02, Las Palmas, Spain (2002)
3. Gildea, D., Jurasfky, D.: Automatic labeling of semantic roles. Computational Linguistic **28**(3) (2002) 496–530
4. Carreras, X., Màrquez, L.: Introduction to the CoNLL-2005 shared task: Semantic role labeling. In proceedings of CoNLL-2005, Ann Arbor, Michigan, (2005)
5. Moschitti, A.: A study on convolution kernels for shallow semantic parsing. In proceedings of ACL'04, Barcelona, Spain (2004)
6. Pradhan, S., Hacioglu, K., Krugler, V., Ward, W., Martin, J.H., Jurafsky, D.: Support vector learning for semantic argument classification. Machine Learning Journal (2005)
7. Moschitti, A., Coppola, B., Pighin, D., Basili, R.: Engineering of syntactic features for shallow semantic parsing. In proceedings of the ACL Workshop on Feature Engineering for Machine Learning in Natural Language Processing, Ann Arbor, Michigan, (2005)
8. Moschitti, A., Pighin, D., Basili, R.: Tree kernel engineering in semantic role labeling systems. In proceedings of the EACL Workshop on Learning Structured Information in Natural Language Applications, Trento, Italy, (2006)
9. Charniak, E.: A maximum-entropy-inspired parser. In: Proceedings of the 1st Meeting of NAACL. (2000)
10. Haghighi, A., Toutanova, K., Manning, C.: A joint model for semantic role labeling. In proceedings of CoNLL-2005, Ann Arbor, Michigan. (2005)
11. Moschitti, A., Pighin, D., Basili, R.: Semantic role labeling via tree kernel joint inference. In proceedings of CoNLL-X. (2006)
12. Moschitti, A.: Efficient convolution kernels for dependency and constituent syntactic trees. In proceedings of ECML 2006, Berlin, Germany. (2006)
13. Joachims, T.: Making large-scale SVM learning practical. In Schölkopf, B., Burges, C., Smola, A., eds.: Advances in Kernel Methods - Support Vector Learning. (1999)

Frequent Subgraph Miners: Runtimes Don't Say Everything

Siegfried Nijssen[1] and Joost N. Kok[2]

[1] Albert-Ludwigs-Universität, Georges-Köhler-Allee, Gebäude 097, D-79110, Freiburg im Breisgau, Germany.
[2] LIACS, Leiden University, Niels Bohrweg 1, 2333 CA, Leiden, The Netherlands.
snijssen@informatik.uni-freiburg.de

Abstract. In recent years several frequent subgraph miners were proposed. The authors of these new algorithms typically compared the runtimes of their implementations with those of previous implementations to confirm the efficiency of their methods. To get a better perspective on the mutual benefits of the algorithms, Wörlein et al. [9] performed an experimental evaluation of re-implementations of several depth-first graph miners, where also some statistics beyond runtimes were compared. In this paper we present results of an additional experimental comparison of several graph miners, which differs in the following aspects from this previous study: (1) we compare original implementations; (2) we compare these implementations on a larger set of measures than runtimes, thus providing further insight in the benefits of the algorithms; (3) we include breadth-first graph miners and free tree miners in the comparison.

1 Introduction

Given a database consisting of small graphs, for example, molecular graphs, the problem of mining frequent subgraphs is to find all subgraphs that are subgraph isomorphic with a large number of example graphs in the database. References to applications can be found in [2, 4, 8, 3]. The first frequent subgraph miner was Inokuchi et al.'s AGM algorithm (2000), for unconnected subgraphs. The algorithm was followed by the FSG algorithm of Kuramochi and Karypis [7] (2001), and an adaption of AGM, AcGM, for mining connected subgraphs [6] (2002). These initial algorithms performed the search breadth-first; the first depth-first graph miners were MoFA [2] and gSpan [10] (2002), followed by FFSM [5] (2003) and Gaston [8] (2004). In parallel, also free tree (cycleless graph) miners were developed: both the breadth-first FreeTreeMiner (2003) and the depth-first HybridTreeMiner (2004) by Chi et al. (see [3] for references). The development of new graph miners was motivated by the supposed inefficiency of earlier graph miners. In this paper, we perform a large set of experiments with the implementations of the graph and free tree miners that were used in most of these original publications.

We have several aims by doing these experiments. First, our study verifies earlier studies, both the studies of the original implementers, as the re-implementation study of [9]. This provides either additional evidence for the

results obtained in these studies, or will allow us to refute claims made in these studies, in both cases providing us additional insights in the effects of, for instance, implementation optimizations. Second, by computing additional measures, we wish to obtain a deeper understanding in the true benefits of the graph mining algorithms. It can be observed that most graph mining algorithms can be separated in separate components for 'candidate generation' and 'candidate evaluation'. Until now, most algorithms provide a unique combination for each of these elements. By performing additional experiments, we can provide insight in the question if it is useful to try out different combinations of these components. Thus, this study is complementary to the earlier graph mining study of [9], and our earlier study on the efficiency of frequent tree miners [3].

The paper is organized as follows. First, we shortly review the problem of frequent subgraph mining and the frequent subgraph miners. The main part of the paper consists of a large number of experiments. Finally, we conclude.

2 Concepts and Algorithms

In this section we briefly recall the most important concepts and algorithms.

Concepts A *labeled undirected graph* is a quadruple (V, E, λ, Σ), where V is a set of vertices, $E \subseteq \{e \subseteq V : |e| = 2\}$ is a set of edges, and $\lambda : V \cup E \rightarrow \Sigma$ is a function that assigns labels to the vertices and edges. In this paper, we only consider connected graphs, in which there is a path between every pair of nodes. Graph $G' = (V', E', \lambda', \Sigma')$ is a subgraph of graph G if $V' \subseteq V$, $E' \subseteq E$, for all $x \in V' \cup E' : \lambda'(x) = \lambda(x)$ and $\Sigma' \subseteq \Sigma$. Graph G and G' are isomorphic iff there is a bijective function $\phi : V \rightarrow V'$ such that (1) $\forall \{v_1, v_2\} \in E : \{\phi(v_1), \phi(v_2)\} \in E'$ and $\lambda(\{v_1, v_2\}) = \lambda'(\phi(\{v_1, v_2\}))$, (2) $\forall v \in V : \lambda(v) = \lambda'(\phi(v))$. Graph G is subgraph isomorphic with G', denoted by $G' \succeq G$, iff G is isomorphic with a subgraph of G'. The corresponding function ϕ, which maps nodes from the subgraph to the supergraph, is called an embedding. Given a multiset of graphs D (also referred to as database graphs), the frequency (or support) of a graph G, denoted by $freq(G)$ is the cardinality of the set $\{G' \in D | G' \succeq G\}$. Assuming that each database graph has an identifier, this set could alternatively consist of identifiers; we refer to this set as the transaction identifier (TID) list of the subgraph. Given a threshold *minsup*, a (pattern) graph G is frequent iff $freq(G) \geq minsup$; frequent subgraph miners find all such subgraphs. An important property is the antimonotonicity property that states that if $G \succeq G'$, $freq(G) \leq freq(G')$. If we start searching from the smallest subgraphs, a consequence of this property is that we do not need to consider supergraphs of infrequent graphs; we can cut parts of the search space.

Two different subgraphs can be isomorphic with each other; let $V = \{v_1, v_2\}$, $E = \{\{v_1, v_2\}\}$ and $\Sigma = \{\mathtt{a}, \mathtt{b}\}$; then the graph $G_1 = (V, E, \lambda_1, \Sigma)$ with $\lambda_1(v_1) = \mathtt{a}$ and $\lambda_1(v_2) = \mathtt{b}$ is isomorphic with the graph $G_2 = (V, E, \lambda_2, \Sigma)$ having $\lambda_2(v_1) = \mathtt{b}$ and $\lambda_2(v_2) = \mathtt{a}$. Care has to be taken that frequent subgraph miners do not find such isomorphic subgraphs multiple times. All algorithms address this issue by determining a canonical string for a graph. The idea is that every

graph has a corresponding string representation, for example, $(v_1, v_2, \mathtt{a}, \mathtt{b})$ for G_1 and $(v_1, v_2, \mathtt{b}, \mathtt{a})$ for G_2. By imposing a total order on the strings of isomorphic graphs, for example, lexicographically, we can decide that one representation is the highest. This string is considered to be the canonical representation of the isomorphic graphs. As no polynomial algorithm is known to compute graph isomorphism or subgraph isomorphism, all graph mining algorithms use exponential search algorithms to find the canonical label or embeddings; free tree mining algorithms, on the other hand, can exploit polynomial algorithms.

Breadth-First Algorithms The breadth-first algorithms have the same setup as the original Apriori algorithm for mining frequent itemsets [1]. They iterate a process of generating candidate subgraphs and determining their support in the database. Candidates are generated by joining two subgraphs that were previously found to be frequent. After the joined subgraph is obtained, it is checked whether all its subgraphs are frequent; if not, the candidate is pruned.

To represent candidates, AcGM and FSG use a canonical string that is obtained from an adjacency matrix. FSG's code allows for the quick computation of the canonical string for any given graph. AcGM's representation is optimized to minimize the generation of non-canonical graphs. Chi et al.'s FreeTreeMiner is similar to AcGM, but uses a polynomially computable canonical string.

The search for embeddings is improved by exploiting the observation that an embedding for a supergraph is also an embedding for its subgraphs. In FSG and the FreeTreeMiner, a TID list is stored with every pattern graph. AcGM takes the idea a step further by also storing one embedding for each database graph in the TID list, and using it as the start of the exponential search.

Depth-First Algorithms The depth-first algorithms do not subdivide the search into strict candidate generation and candidate evaluation phases. Essentially, each of these algorithms scans all embeddings of a subgraph in the database, and collects from that scan the support of the refinements, i.e., the individual edges and nodes that can be connected to the subgraph. The search recurses immediately on each of the frequent refinements.

To avoid duplicate subgraphs in the output, all depth-first graph miners use a special kind of canonical string, which has the property that every prefix of the string is also canonical. A refinement corresponds to extending a canonical string. For each extended string, it is checked whether it is canonical; any non-canonical string can be removed immediately. Although for every refined string it has to be checked with an exponential algorithm if the resulting string is really canonical, an interesting property of most codes is that it often easy to decide that a code is *not* canonical. As a small example, almost all codes sort sibling nodes in the order of their labels. This order limits which new siblings can be added; we will call this the 'label trick'. Such simple tricks limit which nodes in the data have to be scanned for extensions. gSpan, FFSM and Gaston differ in their canonical string, and consequently, also in the 'easy' rules that can be used to eliminate strings that are not canonical:

- gSpan uses a code that consists of a list of edges in the order of a depth-first traversal tree of the subgraph. Canonical refeinements can only connect to a node on the rightmost path of this tree. The 'label trick' applies to gSpan.
- FFSM uses a code that consists of a concatenation of columns of an adjacency matrix. The last column of the adjacency matrix restricts to which nodes new nodes can be connected; 'label tricks' can also be applied in FFSM.
- Gaston uses a code that consists of a concatenation of a code for paths, trees and cycles. The code allows to determine in $O(1)$ time if a refinement leads to a non-canonical code for a tree. The 'label trick' can not be applied.
- The HybridTreeMiner uses another code to represent free trees. The code is efficiently computable, and guarantees that trees or paths are never enumerated more than twice.

To evaluate the frequency of subgraphs, several alternatives have been proposed. In gSpan and Gaston (RE variant) a TID list is maintained with every subgraph. All embeddings are recomputed for graphs in the TID list. For each embedding, refinements that are not pruneable by easy rules, are counted. Non-canonical refinements are pruned later.

An alternative is to not to recompute embeddings, but to store them, as only embeddings of subgraphs can lead to embeddings of supergraphs. Although the frequency of refinements could be collected from the data, FFSM and Gaston (OS variant) use a more elaborate approach, in which the embeddings of some refinements are obtained by *joining* the embeddings of other subgraphs. The motivation is that this reduces the chances of building embedding lists for graphs that are not frequent, as at least two subgraphs need to be frequent before an embedding list is constructed. However, to reduce the number of subgraphs for which the embedding list has to be stored at the same time, it is necessary that joining is performed in a 'local' way in the search tree. To this purpose, all algorithms require an additional set of embedding lists for graphs that are not canonical.

3 Experiments

For our comparison, we obtained source code of the FreeTreeMiner, the HybridTreeMiner and Gaston (RE, OS), and binaries of gSpan, FFSM, FSG and AcGM. The source code was compiled using the GNU C compiler and the O3 compiler option. We used a range of 6 different datasets: (1) three tree datasets (A1, A3, A4) were generated using Zaki's tree dataset generator [11]. A4 is equal to A1, except for the node labels. Most frequent trees in dataset A3 have diameter 2, and differ mainly in degree of the nodes; (2) two molecular datasets. The PTE dataset was used in [10, 8], and includes hydrogens in the graph encoding; the Cancer datasets is a similar, although larger, dataset obtained from the National Cancer Institute, and does not include hydrogen in the encoding[3]; (3) a protein secondary structure dataset of Huan et al. [4].

[3] We performed further experiments with molecular datasets and real-world tree datasets. Results on these datasets are similar and omited due to space constraints.

	A1	A3	A4	PTE	Cancer	Protein
Number of graphs	5000	10000	5000	340	32557	40
Number of nodes	62936	183743	62936	9189	857126	9502
Number of edges	57936	173743	57936	9317	922081	22016
Avg max number equally labeled nodes	2.7	3.9	12.6	11.9	19.2	25.6
Avg number of nodes	12.6	18.4	12.6	27.0	26.3	237.6
Avg max number equally labeled neighbors	1.7	3.2	5.2	2.6	2.7	1.1
Number of node labels	10	10	1	66	67	20
Number of edge labels	1	1	1	4	3	1

Fig. 1. Characteristics of the datasets.

	PTE				Cancer
Algorithm	2%	3%	4%	5%	4%
GASTON (OS)	9.1MB	4.4MB	3.4MB	3.0MB	430MB
GASTON (RE)	1.5MB	1.3MB	1.3MB	1.3MB	23MB
FFSM	8.2MB	4.1MB	3.7MB	3.2MB	257MB
gSpan	3.8MB	2.8MB	2.8MB	2.8MB	46MB
AcGM	33.9MB	5.3MB	2.2MB	1.6MB	434MB
FSG	123.5MB	25.8MB	25.8MB	25.8MB	107MB

Fig. 2. Memory usage of the algorithms.

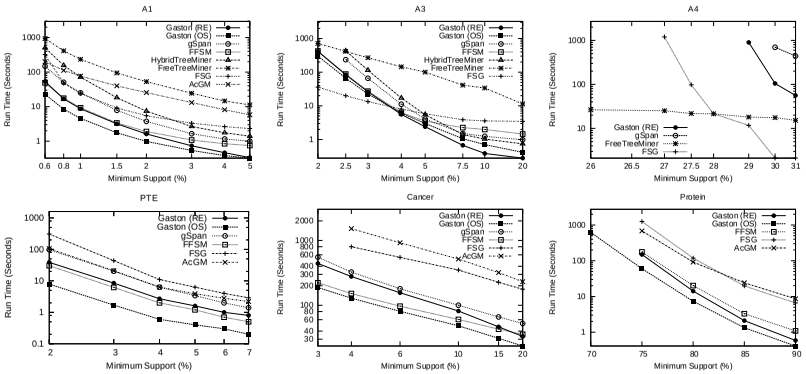


Fig. 3. Runtime experiments on all datasets.

	A1				A3			
Minimum support	30%	10%	8%	6%	7.5%	4%	3%	2.5%
Frequent graphs	838	18552	36945	93979	156	1826	5405	10451
gSpan —								
Suboptimality	103%	107%	108%	108%	107%	101%	101%	101%
FFSM —								
Suboptimality	310%	231%	213%	195%	254%	289%	254%	234%
Join efficiency	99%	70%	62%	55%	100%	99%	91%	85%
GASTON —								
Suboptimality	207%	136%	125%	115%	212%	214%	161%	136%
Join efficiency	27%	41%	43%	47%	11%	29%	44%	52%
Join necessity	97%	95%	96%	97%	100%	98%	98%	99%
HYBRIDTREEMINER —								
Suboptimality		160%	168%	173%		104%	101%	101%
Join efficiency		21%	20%	21%		33%	39%	42%
Join necessity		96%	96%	94%		100%	100%	100%

	PTE				Protein			Cancer		
Minimum support	20%	6%	3%	2%	80%	75%	70%	9%	5%	3%
Frequent graphs	190	2326	22758	136949	26740	276297	2709331	1818	5663	15566
gSpan —										
Suboptimality	139%	134%	137%	129%	-	-	-	161%	158%	159%
FFSM —										
Suboptimality	192%	152%	133%	134%	166%	144%	132%	-	-	-
Join efficiency	61%	50%	45%	41%	38%	35%	35%	-	-	-
GASTON —										
Overall suboptimality	176%	149%	151%	165%	180%	219%	241%	170%	157%	147%
Free tree suboptimality	160%	125%	109%	103%	129%	111%	104%	156%	139%	126%
Cyclic join efficiency	100%	100%	94%	89%	70%	75%	80%	39%	29%	28%
Cyclic join necessity	100%	100%	94%	90%	82%	68%	61%	94%	93%	93%
Free tree join efficiency	52%	37%	64%	75%	38%	49%	55%	27%	27%	27%
Free tree join necessity	83%	73%	86%	90%	100%	99%	99%	96%	95%	95%

Fig. 4. Characteristics of the gSpan, FFSM and GASTON algorithms on the A1, A3, PTE, Protein and Cancer datasets.

Properties regarding the number of equally labeled nodes, edges and neighbors, are listed in Figure 1. These properties are important, as they yield a higher branching factor in algorithms that perform an exponential search for embeddings. The datasets represent a broad range of such properties.

To perform our experiments we used two computers: all cyclic graph datasets were mined on an AMD Athlon XP1600+ with 512MB main memory, running Mandrake Linux 10; all free tree datasets were mined on an Intel Pentium IV 2.8Ghz with 512MB main memory, running Red Hat Linux 7.3.

The results of runtime experiments are given in Figure 3. The experiments are measured using the Unix `time` command and are averaged over 3 runs.

The experiments show the importance of the branching factor of subgraph isomorphism algorithms. The depth first graph mining algorithms fail miserably on the A4 dataset, which has a high branching factor (they run too long, or consume too much memory); only the breadth-first FreeTreeMiner performs well. Otherwise, however, the FreeTreeMiner usually performs worse than FSG. The A3 dataset shows the difference between breadth-first algorithms that have to find one embedding per database graph (like FSG), and depth-first algorithms that have to find all embeddings. In some cases the breadth-first graph miners run out of memory, which is caused by extremely large sets of candidates.

The results on the cyclic datasets are similar to those on the tree datasets, and confirm the observations of previous publications.

The runtime experiments however do not show whether differences are due to a 'better' canonical form, as claimed by several authors, or due to more optimized implementations or memory-runtime trade-offs. Results of experiments to assess the quality of the canonical form are listed in Figure 4. The following statistics are listed in this table:

- frequent graphs: the number of frequent subgraphs resulting from the runs of all algorithms;
- suboptimality: the number of frequent subgraphs for which the canonical string test is computed, divided by the number of frequent subgraphs; the higher this number is, the less well does the code prevent isomorphic graphs using easy rules, and thus more redundant supports are computed;
- join efficiency: the number of joins that results in a frequent subgraph, divided by the total number of joins that is performed; the closer to 100% this value is, the better does the algorithm succeed in only computing embedding lists that are really frequent; in the case of Gaston, there is a distinction between the joins between embedding lists of trees and of cyclic graphs;
- join necessity: the number of joins that results in a subgraph with support higher than zero, divided by the total number of joins that is performed; if this number would not be close to 100%, joining of embedding lists would not make much sense, as we are performing many computations that an algorithm that collects extensions from data would not need to perform.

For Gaston and the HybridTreeMiner, we obtained these measures by changing the source code. For gSpan and FFSM, we used the Valgrind tool available

	A1 1%				PTE 2%				Protein 75%		
Algorithm	1×	3×	5×	Regression	1×	2×	3×	Regression	1×	2×	3×
Gaston (OS)	4.6s	14.8s	26.1s	5.5x-1.4s	7.9s	15.2s	23.0s	7.5x+0.4s	60s	116s	183s
Gaston (RE)	8.7s	27.7s	48.0s	9.9x-1.8s	39.9s	76.3s	112.4s	36.2x3.7s	148s (78s)	398s (166s)	643s (290s)
FFSM					29.7s	55.7s	82.2s	26.3x+3.4s	177s (148s)	353s (398s)	554s (643s)
gSpan	15s	47s	81s	16.6x-2.2s	100s	186s	271s	85.4x+14.9s			
FSG	17s	35s	53s	9.1x+7.7s	316s	402s	489s	86.3x+229.8s	1253s	2264s	3275s
AcGM					107s	170s	234s	63.5x+43.3s			
HybridTreeMiner	46s	146s	248s	48.9x-1.1s							
FreeTreeMiner	243s	715s		238.9x+1.5s							

Fig. 5. Scale-up experiments on several datasets; experiments between brackets were obtained on an Intel Pentium IV.

at `www.valgrind.org`. Valgrind makes it possible to count numbers of function calls, even with sufficient reliability for binaries that have been compiled without debugging information. We counted the number of calls to the `isCanonicalForm` (FFSM) and `isDuplicate` (gSpan) functions, allowing us to determine the sub-optimality of the graph codes. The functionality of these functions was confirmed by the authors of the binaries.

The tables provide a large amount of information. Most interesting is the good performance of gSpan's code. In terms of optimality, this code performs best in almost all cases. This result suggests that if we combine the code of gSpan with the evaluation mechanisms of the other algorithms, we might achieve similar runtimes as for these other algorithms. Comparing Gaston to FFSM, Gaston is less optimal than FFSM on cyclic graphs, but joins more efficiently. Only considering trees in the molecular databases, Gaston's code is more optimal. The good result of the HybridTreeMiner on the A3 dataset can be explained by the fact that most frequent subtrees in this database turn out to be (single) centred; HybridTreeMiner's tree code is optimized for such trees.

The most interesting observation is that there is no strong relation between the runtime experiments and the efficiencies of the graph codes. Then what does determine the efficiency of the graph miners?

More insights can be obtained from the experiments in Figure 5 and Figure 2. In Figure 5, we repeat the same mining experiments for increasingly larger datasets that are obtained by concatenating the same dataset multiple times. In general, the algorithms scale linearly. The constant in the linear regression relates to the operations that are independent of the dataset size, such as candidate generation. The time to count candidates is comparable in both the breadth-first and depth-first graph miners that do not use embedding lists. The effort to find all embeddings is apparently negligibly higher than the effort to find only one. On the Protein dataset, however, the scale-up is not linear. If we perform this experiment on two different computers, we can conclude that he reason is the very small size of this dataset. Without using embedding lists, the data fit all within the cache of the CPU, and the computation is extremely

fast. On a computer with less cache (like the Athlon) the advantage of in-cache computations is lost more quickly as the dataset grows larger.

Finally, the memory usage experiments show that the runtime differences between FFSM and gSpan are the result of a memory-CPU trade-off. Please note that the differences in memory usage are strongly influenced by the choices made in the binaries for the amount of bits used to represent node labels, etc., and are therefore only indicative.

4 Conclusions

In this study, we have confirmed that there is not a strong relation between the canonical string that is used and the runtime behavior of the algorithms. Polynomially computable codes, such as used in frequent tree mining, are often less efficient from a practical point of view. Of the many canonical codes that have been proposed, the DFS code that was introduced in gSpan perform consistently most well. Given its conceptual simplicity, this code should be preferred.

The differences between breadth-first and depth-first graph mining are significant. In most cases, the depth-first miners are faster and require less memory.

Many of the runtime differences between graph miners can be attributed to two aspects. First, there are the typical high performance computing issues, such as caching behavior; second, there is the memory usage-runtime trade-off.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328, 1996.
2. C. Borgelt and M. R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *ICDM*, pages 51–58, 2002.
3. Y. Chi, S. Nijssen, R. R. Muntz, and J. N. Kok. Frequent subtree mining—An overview. In *Fundamenta Informaticae*, volume 66, pages 161–198, 2005.
4. J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha. Mining family specific residue packing patterns from protein structure graphs. In *RECOMB*, pages 308–315, 2004.
5. J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM*, pages 549–552, 2003.
6. A. Inokuchi, T. Washio, K. Nishimura, and H. Motoda. A fast algorithm for mining frequent connected subgraphs. Technical Report RT0448, IBM Research, 2002.
7. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM*, pages 313–320, 2001.
8. S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *KDD*, pages 647–652, 2004.
9. M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen. A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and Gaston. In *PKDD*, LNCS 3721, pages 392–403, 2005.
10. X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.
11. M. J. Zaki. Efficiently mining frequent trees in a forest. In *KDD*, pages 71–80, 2002.

Graph Kernels versus Graph Representations: a Case Study in Parse Ranking

Tapio Pahikkala, Evgeni Tsivtsivadze, Jorma Boberg, and Tapio Salakoski

Turku Centre for Computer Science (TUCS)
Department of Information Technology, University of Turku
Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland
firstname.lastname@it.utu.fi

Abstract. Recently, several kernel functions designed for a data that consists of graphs have been presented. In this paper, we concentrate on designing graph representations and adapting the kernels for these graphs. In particular, we propose graph representations for dependency parses and analyse the applicability of several variations of the graph kernels for the problem of parse ranking in the domain of biomedical texts. The parses used in the study are generated with the link grammar (LG) parser from annotated sentences of BioInfer corpus. The results indicate that designing the graph representation is as important as designing the kernel function that is used as the similarity measure of the graphs.

1 Introduction

Structured data are a commonplace in areas such as natural language processing (NLP). One of the most frequently encountered data structures in NLP are graphs. Kernel methods (see e.g. [1, 2]) have been among the most successful and computationally effective learning algorithms that can take advantage of the structured representation of the data. Recently, kernel functions on instances that are represented by graphs were introduced by [3–7]. Inspired by this research, we propose graph representations for dependency parses and analyse the applicability of the graph kernels for the problem of parse ranking in the domain of biomedical texts.

The link grammar (LG) parser [8] used in our research is a full dependency parser based on a broad-coverage hand-written grammar. The parses are generated by the LG parser applied to BioInfer corpus [9] containing 1100 annotated sentences. Due to the complexity of the biomedical text, the number of parses generated per sentence is large. Recently, we introduced a method for dependency parse ranking [10] that uses regularized least-squares (RLS) algorithm [11] and grammatically motivated features. The method, called RLS ranker, worked notably better giving 0.42 correlation compared to 0.16 of the LG heuristics measured with the Kendall's τ_b correlation coefficient [12]. In [13], we further developed the method by designing nonlinear kernel functions suitable for the problem.

In this study, we concentrate on designing graph representations, which is often overlooked. We demonstrate that designing an appropriate graph representation has a notable influence on the final results, comparable to the influence of the kernel function. RLS using graph kernels is applied to the proposed graph representations and the results indicate an improved correlation of 0.45. A detailed description of the data, RLS algorithm and the experimental setup used in this paper is given in [10]. We also show that the proposed approach can be considered as a generalization over previously described method [10].

2 Graph Kernels

We now give a brief introduction to kernels considered in this study. Formally, let X denote the input space, which can be any set, and H denote the feature space. For any mapping $\Phi : X \to H, k(x, z) = \langle \Phi(x), \Phi(z) \rangle$ is a kernel function. Following [3], we define a labeled graph representation of data points as follows. Below, $\mathcal{M}_{i \times j}(\mathbb{R})$ denotes the set of real valued matrices of dimension $i \times j$ and $[M]_{i,j}$ denotes the element of matrix M in the i-th row and j-th column. Let $\mathcal{L} = \{l\}_r, r \in \mathbb{N}^+$ be an enumeration of all possible labels. Let $G = (V, E, h)$ be a graph that consists of the set of vertices V, the set of edges $E \subseteq V \times V$, and a function $h : V \to \mathcal{L}$ that assigns a label to each vertex of a graph. We assume that the edge set of G is represented as an adjacency matrix $A \in \mathcal{M}_{|V| \times |V|}(\mathbb{R})$ whose rows and columns are indexed by the vertices V, and $[A]_{i,j}$ is one if the vertices $v_i \in V$ and $v_j \in V$ are connected with an edge, and zero otherwise. We also assume that the function h is represented as a label allocation matrix $L \in \mathcal{M}_{|\mathcal{L}| \times |V|}(\mathbb{R})$ so that its element $[L]_{i,j}$ is one if vertex $v_j \in V$ has a label $l_i \in \mathcal{L}$ and zero otherwise.

We also define the following relaxed version of the graph labeling. Let $\mathcal{L}$ be a vector space, for example, $\mathcal{L} = \mathbb{R}^p$. We can then define $h : V \to \mathcal{L}$ to be a function that assigns a label vector to each vertex of a graph. Its corresponding representation as a label allocation matrix is $L \in \mathcal{M}_{|\mathcal{L}| \times |V|}(\mathbb{R})$ so that the columns of the matrix are the label vectors of the vertices.

Again, we follow [3], and define a class of kernel functions on labeled graphs. Let us consider the nth power A^n of the adjacency matrix of the graph G. Then, $[A^n]_{i,j}$ is the number of walks of length n from vertex v_i to vertex v_j. When we take the labels of the vertices into account, we observe that $[LA^nL^{\mathrm{T}}]_{i,j}$ is the number of walks of length n between vertices labeled l_i and l_j. Let G and G' be labeled directed graphs and let $\langle M, M' \rangle_F$ denote the Frobenius product of matrices M and M', that is, $\langle M, M' \rangle_F = \sum_{i,j} [M]_{i,j}[M']_{i,j}$. Let further $\gamma \in \mathcal{M}_{n \times n}(\mathbb{R})$ be a positive semidefinite matrix whose eigen decomposition is $U \Lambda U^{\mathrm{T}}$, where U is a matrix that contains the eigenvectors of γ and Λ is a diagonal matrix containing the eigenvalues of γ. We define the kernels k_n between the graphs G and G' as follows

$$\begin{aligned} k_n(G, G') &= \sum_{i,j=0}^{n} [\gamma]_{i,j} \langle LA^iL^{\mathrm{T}}, L'A'^jL'^{\mathrm{T}} \rangle_F \\ &= \sum_{k,l=1}^{|\mathcal{L}|} \sum_{i,j=0}^{n} [\gamma]_{i,j} [LA^iL^{\mathrm{T}}]_{k,l} [L'A'^jL'^{\mathrm{T}}]_{k,l} \\ &= \sum_{k,l=1}^{|\mathcal{L}|} \sum_{i=0}^{n} \phi_{i,k,l}(G) \phi_{i,k,l}(G') \end{aligned} \tag{1}$$

where it is easy to see that, when γ is positive semidefinite, the features are defined as $\phi_{i,k,l}(G) = \sum_{j=0}^{n}[\sqrt{\Lambda}U^{\mathrm{T}}]_{i,j}[LA^jL^{\mathrm{T}}]_{k,l}$. By specializing $[\gamma]_{i,j}$ in (1), we obtain several kernel functions with different interpretations. For example, if we set $[\gamma]_{i,j} = \theta^i\theta^j$, where $\theta \in \mathbb{R}^+$ is a parameter, we obtain the kernel

$$\widehat{k_n}(G, G') = \langle L(\sum_{i=0}^{n}\theta^i A^i)L^{\mathrm{T}}, L'(\sum_{i=0}^{n}\theta^i A'^i)L'^{\mathrm{T}}\rangle_F. \tag{2}$$

This kernel can be interpreted as an inner product in a feature space in which there is a feature $\phi_{k,l}$ per each label pair (k, l) so that its value $\phi_{k,l}(G)$ for a graph G is a weighted count of walks of length up to n from the vertices labeled l to the vertices labeled k. On the other hand, by setting $[\gamma]_{i,j} = \theta^i$ when $i = j$ and zero otherwise, we obtain the kernel

$$\widetilde{k_n}(G, G') = \sum_{i=0}^{n}\theta^{2i}\langle LA^iL^{\mathrm{T}}, L'A'^iL'^{\mathrm{T}}\rangle_F, \tag{3}$$

which can be interpreted as an inner product in a feature space in which there is a feature $\phi_{i,k,l}$ per each tuple (i, k, l), where l and k are labels and i is a length of a walk. Its value $\phi_{i,k,l}(G)$ for a graph G is θ^i times the count of walks of length i from the vertices labeled l to the vertices labeled k. Finally, with certain conditions on the coefficients $[\gamma]_{i,j}$, we can also define $k_\infty(G, G') = \lim_{n\to\infty} k_n(G, G')$. One such kernel function is, for example, the exponential graph kernel $k_{exp}(G, G') = \langle Le^{\beta A}L^{\mathrm{T}}, L'e^{\beta A'}L'^{\mathrm{T}}\rangle_F$, where β is a parameter and $e^{\beta A}$ can be written as $e^{\beta A} = \lim_{n\to\infty}\sum_{i=0}^{n}\frac{\beta^i}{i!}A^i$. In this case, the coefficients $[\gamma]_{i,j}$ are determined by the parameter β as follows: $[\gamma]_{i,j} = \frac{\beta^i}{i!}\frac{\beta^j}{j!}$.

2.1 Graph Representations of the Parses

The output of the LG parser contains the following information for each input sentence. The linkage consisting of pairwise dependencies between pairs of words termed links. An example of a parsed sentence is presented in Fig.1. In addition to the linkage structure, the parses contain information about the link types (the grammatical roles assigned to the links) used to connect word pairs. The link types present in Fig.1 are *Mp*, *Js*, *Ss*, etc. Further, the parse contains the part-of-speech (PoS) tags of the words, such as verb (v), noun (n) and adjective (a) categories. In Fig.1, the assigned PoS tags, for example, to the words *absence*, *alpha-syntrophin*, *leads* are *n*, *n v*, respectively. Different parses of a single sentence have a different combination of these elements.

In our previous study [10], we proposed several grammatically motivated features that could be extracted from the above described dependency parses, namely the link type, link length, PoS, word & link type, word & PoS, link type & link length, grammatical bigram, and link bigram features that we will describe below in more detail. In this paper, we are using graph kernels as similarity measures of the data points. Therefore, we now define a graph representation for the parses. The representation is designed so that we are able to simulate the previously proposed (and some additional) features with the graph kernel.

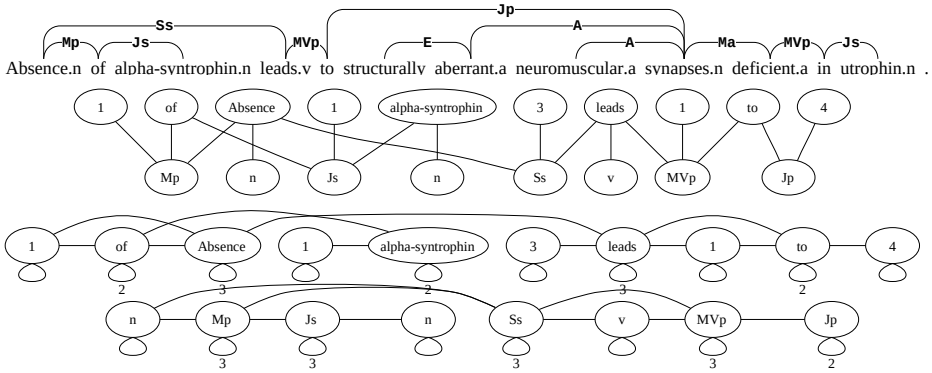


Fig. 1. Example of parsed sentence (top). Graph representation of the beginning of the parse (middle). Walks of length 2 present in the parse (bottom). Note that some of the loops (the edges starting from and ending to the same node) have weights larger than one, that is, there are several repetitions of the corresponding walks.

Graph representation. Let p be a parse generated from a sentence s and let $G = (V, E, h)$ denote the graph representation of p. An example of the graph representation is presented in Fig.1. Let us first define the vertices of p. For each word in the sentence s, there is a corresponding vertex $v \in V$ and the vertex is labeled with the word (the word vertices and their labels do not depend from the parse). Thus, if a word occurs several times in the sentence s, all of the occurrences have their own vertices in the graph but the labels of the vertices are equal. For each link in the parse p, there is a corresponding vertex in V that is labeled with its link type. Similarly to the word vertices, a parse may have several occurrences of the same link type. In p, each word is assigned a PoS, for which there is a corresponding vertex in V labeled with the PoS. Further, each link in p has a length (the number of words that a link in the sentence spans) for which there is a corresponding vertex in V labeled with the length. In Fig.1, they are the vertices labeled with integers, for example, *1*, *1*, *3*, etc.

The edges of G are defined as follows. A word vertex and its corresponding PoS vertex are connected with an edge. A link vertex and its corresponding length vertex are connected with an edge. If two words are connected with a link in the parse p, the corresponding link vertex is connected with an edge to both of the corresponding word vertices. The connection of a word vertex in the graph with a link vertex (for example in Fig.1: *absence—Mp*, *absence—Ss*, *of—Js*, etc.) can be considered as the word & link type feature described in our previous study. Below, we show how these connections are used to create also word bigram and link bigram described previously [10].

Random walk features. Let $G = (V, E, h)$ be a graph representation of a parse. Let us consider the second power of the adjacency matrix A of G, that is, the walks of length 2 in the graph representation of the corresponding parse.

Those walks in the graph representation are illustrated in Fig.1. Note that intersection of the sets of the walks of length one and two is empty due to the bipartite property of the graph. Also because of this property, all of the walks of length 2 have both the start and the end vertices in the same subset. Among the walks of length two there is, for example, a walk between two word vertices iff they are connected with a link in the parse, and between two link vertices iff the links are connected to the same word in the parse. Such connections were called grammatical bigrams and link bigrams in [10]. We also obtain walks between between PoS vertices and link vertices, and between word vertices and link length vertices. Finally, a vertex has as many cycles as there are edges connected to it. If we consider the higher powers of the adjacency matrices, we obtain new features, for example, link length pairs in the fourth power. In the higher powers, we also obtain word and link bigrams, where the words and links are not connected to each other in the parse.

Vector labeled graph representation. Using multiple labels for vertices offers a way to incorporate more information into the graph representation. This kind of representations have been considered by [6], for example. In addition to the graph representation presented above, we define the following vector labeled representation. Again, let p be a parse generated from a sentence s and let $G = (V, E, h)$ denote the graph representation of p. For each word in the sentence s, there is a corresponding vertex $v \in V$ and the vertex is labeled with the word and its PoS. In other words, all the elements of the label vector are zero except the ones indexed by the word and its PoS. For each link in the parse p, there is a corresponding vertex in V that is labeled with its link type and its length. Thus, instead of having the PoS and link length vertices as in the previous graph representation of parses, they are used as vertex labels in this representation. We refine the representation further using the following five labels for the link length instead of just one. Let l be the length of a link. Instead of using only l, we use $l-2$, $l-1$, l, $l+1$, $l+2$. Using this set of labels, we are able to match similarities between links whose lengths are close to each other, while the single label is useful only for exact matching of the link lengths. This representation has certain connections that the previous one does not have, such as the connections between word and link length as well as between PoS and link type. On the other hand, it misses the connections between word and PoS as well as between link type and link length that are in the previous representation.

When we consider the walks of length 2, that is, the label connections obtained using the second power of the adjacency matrix, we observe that the missing connections between word and PoS, for example, are among those walks. Note, however, that there are also connections between words and the PoS of their neighboring words, and there is no way to distinguish between these connections and the connections between words and their own PoS. The same problem arises from the connections between link type and link length that are also among the walks of length 2.

3 Experiments

We give a detailed description of the problem and the data in [10]. We evaluate the different variations of the graph kernel by performing a 10-fold cross-validation on the sentence level so that all the parses generated from a sentence would always be in the same fold (see [14] for a fast n-fold cross-validation algorithm for RLS). The RLS algorithm has the regularization parameter λ that controls the trade-off between the minimization of the training error and the complexity of the regression function. The appropriate values of this parameter is determined together with the kernel parameters by grid search with 10-fold cross-validation.

In our experiments, we evaluate the kernels $\widehat{k_n}$ and $\widetilde{k_n}$ up to the third power with different parameters. In addition, we evaluate the following version of the exponential graph kernel $\bar{k}_{exp}(G, G') = \langle LAe^{\beta A}L^{\mathrm{T}}, L'A'e^{\beta A'}L'^{\mathrm{T}}\rangle_F$. We multiply the exponentiated adjacency matrix $e^{\beta A}$ by A, because then by setting $\beta = 0$, we obtain the original adjacency matrix A as a special case, and we can set the preferred weight of the higher powers by a grid search on β.

We start by evaluating the graph kernel k_0 with $[\gamma]_{0,0} = 1$, that is, the zeroth power of the adjacency matrix. The obtained performance of the RLS ranker with k_0 is 0.377. This corresponds to a kernel that counts the elementary features that are present in both parses, that is, the words, link types, PoS tags, and link lengths. Recall that the word vertices only depend on the sentence, and therefore they are useless for the parse ranking. In our previous study [10], we also found out that the other elementary features are not as good as the combination features. We continue by evaluating k_1 with $[\gamma]_{i,j} = 1$ when $i = j = 1$ and zero otherwise. In other words, we use only the original adjacency matrices of the graphs determined by the edges defined in Sect.2. The result of this experiment is 0.406 correlation points. The performance differences in these two experiments are in correspondence to our previous study [10], in which we observed that the ranking performances with the four elementary features were low compared to their combinations that are present in the first power of the adjacency matrices.

The second powers of the adjacency matrices contain the elementary features present in the zeroth power, the rest of the combination features proposed in the previous study that are present in the first and the second powers, and also some new combination features in the second power. To get a weighted combination of the first and the second power features, we evaluate the following kernel function

$$k(G, G') = \langle L(A + \theta A^2)L^{\mathrm{T}}, L'(A' + \theta A'^2)L'^{\mathrm{T}}\rangle_F, \tag{4}$$

where θ is a parameter for which we perform a grid search in range $2^{-5}, 2^{-4}, \ldots, 2^5$. The above kernel is equal to $\widehat{k_2}$ in (2) except that we exclude the zeroth power. Note that due to the bipartite property, the kernel is also equal to $\widetilde{k_2}$ in (3) with the zeroth power excluded. The performance with the best θ parameter is 0.429 correlation points.

To analyse the usefulness of the walk features of length longer than 2, we evaluate the following two kernels

$$k(G, G') = \langle L(A + \theta A^2 + \theta^2 A^3)L^{\mathrm{T}}, L'(A' + \theta A'^2 + \theta^2 A'^3)L'^{\mathrm{T}} \rangle_F, \quad (5)$$

and

$$\begin{aligned} k(G, G') = \langle LAL^{\mathrm{T}}, L'A'L'^{\mathrm{T}} \rangle_F + \theta \langle LA^2L^{\mathrm{T}}, L'A'^2L'^{\mathrm{T}} \rangle_F \\ + \theta^2 \langle LA^3L^{\mathrm{T}}, L'A'^3L'^{\mathrm{T}} \rangle_F, \end{aligned} \quad (6)$$

where the best θ parameters are found with a grid search. The first kernel is similar to $\widehat{k_3}$ in (2) with the zeroth power excluded, and the second kernel is similar to $\widetilde{k_3}$ in (3) with the zeroth power excluded. The performance using the first and the second kernels with the best parameters are 0.422 and 0.429 correlation points, respectively. We also performed some experiments with the fourth powers of the adjacency matrices but the results were worse than the above two. The performance with the exponential graph kernel $\bar{k}_{exp}(G, G')$ is 0.425 correlation points. According to the results, is seems that the walks of length larger than 2 are not useful features when this kind of graph representation is used. In fact, they seem to be even harmful when they are mixed with the lower order features, for example, in the kernel (5).

We also evaluate the vector labeled graph representation presented in Sect.2. The results with the original adjacency matrix and with the kernel (4) are 0.364 and 0.377, respectively. They are clearly worse than the ones with the single labeled graph representation and therefore we do not conduct more experiments with the vector labeled representation. The low performance is probably due to the noisy combination features that we discuss in Sect.2.1. The results indicate that designing the graph representation is as important as designing the kernel function.

In order to validate the results with an unseen test set, we conduct a final validation experiment on 600 sentences reserved for that purpose. We select the single labeled graph representation and the kernels (4) and (6) with the best performing parameter combinations, that is, the settings that give the highest ranking performance in the parameter estimation experiments. The ranker is trained with the parameter estimation data and the results with the kernels (4) and (6) are 0.444 and 0.447 correlation points, respectively.

4 Conclusion

We propose graph representations for dependency parses and analyse the applicability of several variations of the graph kernels for the problem of parse ranking in the domain of biomedical texts. We use the graph kernels to generate features that are based on the start and end labels of random walks between vertices in the graphs. The feature vector corresponding to a data point is determined by its graph representation and the kernel function. Both of them have to be selected carefully in order to ensure a good performance of the machine learning method. Several kernel functions have already been proposed for a data

that consists of graphs. In addition, our results underline the importance of the design of a good graph representation for the data points. The performance achieved is promising when compared to our previous studies [10] and could be even further improved by designing representations capturing additional prior knowledge about the problem to be solved.

Acknowledgments

This work has been supported by Tekes, the Finnish Funding Agency for Technology and Innovation. We would like to thank CSC, the Finnish IT center for science, for providing us extensive computing resources and the anonymous reviewers for their insightful comments.

References

1. Schölkopf, B., Smola, A.J.: Learning with kernels. MIT Press (2002)
2. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press (2004)
3. Gärtner, T.: Exponential and geometric kernels for graphs. In: NIPS Workshop on Unreal Data: Principles of Modeling Nonvectorial Data. (2002)
4. Gärtner, T., Flach, P.A., Wrobel, S.: On graph kernels: Hardness results and efficient alternatives. In Schölkopf, B., Warmuth, M.K., eds.: COLT'03, Springer (2003) 129–143
5. Kashima, H., Inokuchi, A.: Kernels for graph classification. In: ICDM Workshop on Active Mining. (2002)
6. Suzuki, J., Sasaki, Y., Maeda, E.: Kernels for structured natural language data. In Thrun, S., Saul, L.K., Schölkopf, B., eds.: NIPS 16, MIT Press (2003)
7. Vishwanathan, S., Smola, A.J., Vidal, R.: Binet-cauchy kernels on dynamical systems and its application to the analysis of dynamic scenes. International Journal of Computer Vision (2006) To appear.
8. Sleator, D.D., Temperley, D.: Parsing english with a link grammar. Technical Report CMU-CS-91-196, Carnegie Mellon University, Pittsburgh, PA (1991)
9. Pyysalo, S., Ginter, F., Heimonen, J., Björne, J., Boberg, J., Järvinen, J., Salakoski, T.: Bioinfer: A corpus for information extraction in the biomedical domain (2006) Submitted.
10. Tsivtsivadze, E., Pahikkala, T., Pyysalo, S., Boberg, J., Mylläri, A., Salakoski, T.: Regularized least-squares for parse ranking. In Famili, A.F., Kok, J.N., Peña, J.M., Siebes, A., Feelders, A.J., eds.: IDA'05, Springer (2005) 464–474
11. Poggio, T., Smale, S.: The mathematics of learning: Dealing with data. Notices of the American Mathematical Society (AMS) **50**(5) (2003) 537–544
12. Kendall, M.G.: Rank Correlation Methods. 4. edn. Griffin (1970)
13. Tsivtsivadze, E., Pahikkala, T., Boberg, J., Salakoski, T.: Locality-convolution kernel and its application to dependency parse ranking. In Ali, M., Dapoigny, R., eds.: IEA-AIE'06, Springer (2006) 610–618
14. Pahikkala, T., Boberg, J., Salakoski, T.: Fast n-fold cross-validation for regularized least-squares. In: Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006). (2006) To appear.

The Kingdom-Capacity of a Graph: On the Difficulty of Learning a Graph Labeling

Kristiaan Pelckmans[1], Johan A.K. Suykens[1], and Bart De Moor[1]

K.U. Leuven, ESAT, SCD/SISTA, Kasteelpark Arenberg 10, Leuven, B-3001, Belgium
e-mail: kristiaan.pelckmans@esat.kuleuven.be
WWW home page: http://www.esat.kuleuven.ac.be/scd/

Abstract. This short paper[1] establishes a measure of capacity of a fixed graph which can be used to characterize the complexity of learning labels of a graph.[2] We denote it as the Kingdom-capacity (K-capacity) of a graph. It is shown how this notion is implied by the definition of an out-of-sample extension rule predicting the labels of unobserved nodes. An efficient way to compute the K-capacity of a given graph is derived, based on the analogy with a simple strategy game. It is shown how this measure of capacity can be used to construct a nontrivial generalization bound for transductive inference of the labels of the nodes of a given graph.

The aim of learning over graphs received increasing attention in recent years, see e.g. [3]. Such tasks can often be abstracted in terms of associating a $+1$ or -1 label to the respective nodes. Such problems can be related to the task of looking for a minimal cut and variations, see e.g. [11]. Capacity concepts are commonly used in graph theory, they mostly focus on issues of connectedness or are based on maximal matching or coloring results. The Shannon capacity for example characterizes the maximal information which can be transmitted through a network. An efficient SDP relaxation was devised, see e.g. [7]. This work aims at a similar result in the context of learning over graphs.

Therefor, a capacity measure is defined and analyzed, characterizing the complexity of learning the labels of a fixed graph. This notion is analoguous to results of maximal margin classifiers, and can directly be related to the VC dimension in learning theory [6, 12]. The naming convention is intentionally kept different to avoid confusion with *the* VC-dimension of a graph which is defined differently, see e.g. [5].

The Kingdom-Capacity (K-capacity) expresses how many nodes can choose their own label at will ('are king to a neighborhood'). Although this capacity is combinatorial in nature (and hence difficult to compute), it is indicated how a nontrivial upper- and lower-bound respectively can be computed efficiently. More specifically, linear programming problems are formulated yielding such a bound. The formulation of those problems is explained by analogy to a simple strategy game in which one determines the maximal number of king nodes who can all defend their territory (potentially) successfully. This notion can be directly related to capacity concepts of networks, see e.g. [1].

[1] (KP): BOF PDM/05/161, FWO grant V4.090.05N; - (JS) is an associate professor and (BDM) is a full professor at K.U.Leuven Belgium, respectively. (SCD:) GOA AMBioRICS, CoE EF/05/006, (FWO): G.0407.02, G.0197.02, G.0141.03, G.0491.03, G.0120.03, G.0452.04, G.0499.04, G.0211.05, G.0226.06, G.0321.06, G.0553.06, (ICCoS, ANMMM, MLDM); (IWT): GBOU (McKnow), Eureka-Flite2 IUAP P5/22,PODO-II,FP5-Quprodis; ERNSI;

[2] We like to thank professor J. Shawe-Taylor and the anonymous reviewers for constructive remarks.

The following Section formulates this capacity, and motivates this capacity in learning theory. Section 3 discusses a practical approach to efficiently assess this measure of complexity. Section 4 discusses a numerical example, and reviews some further consequences of this line of thought.

1 GRAPH CUTS AND HYPOTHESIS SET

Let a graph G be defined as a set of nodes $\mathcal{V} = (v_1, \ldots, v_n)$ and corresponding edges $\{w_{ij} \geq 0 \text{ between } v_i \text{ and } v_j\}_{i,j=1}^n$. Let the nodes have a corresponding label $\{q_i \in \{-1,1\}\}_{i=1}^n$. Let the vector of positive weights w_i be defined as $(w_{i1}, \ldots, w_{in})^T \in \mathbb{R}_+^n$. Define the neighborhood function $f(v_*)$ of any node v_* as

$$f(v_*) = \sum_{j=1}^{n} w_{*j} q_j = w_*^T q, \tag{1}$$

where $q = (q_1, \ldots, q_n)^T \in \{-1,1\}^n$. Then the neighborhood rule becomes '$\text{sign}(f(v_*))$' as suggested e.g. in [3, 4]. Label y_i is defined to be consistent with its prediction rule if

$$q_i f(v_i) = q_i \sum_{j=1}^{n} w_{ij} q_j = q_i (w_i^T q) \geq 0, \tag{2}$$

and consistent with a margin $\rho > 0$ if $q_i(w_i^T q) \geq \rho$. The set of different hypothical labellings $q \in \{-1,1\}^n$ where each labeling is consistent with corresponding neighborhood rules with a certain margin $\rho > 0$ is thus defined as

$$H_\rho = \left\{ q \in \{-1,1\}^n \;\middle|\; q_i(w_i^T q) \geq \rho, \;\; \forall i = 1, \ldots, n \right\}. \tag{3}$$

Let $|H_\rho|$ denote the cardinality of the set H_ρ. It is seen intuitively that this mechanism of self-consistency may be used to reduce the number of permitted labels: $|H_\rho|$ can be expected to be smaller than 2^n. Before making this statement more precise, a motivation for this restriction of the hypothesis set is given for the case of transductive learning graph labels.

1.1 TRANSDUCTIVE INFERENCE ON GRAPHS

We show how the rate of learning of labels of the deterministic graph G can be expressed in terms of (a bound on) the cardinality of this set. Therefor, the setting of transductive learning is adopted: let a graph $\mathcal{G} = (\mathcal{W}, \mathcal{V})$ with n nodes $\mathcal{V} = \{v_1, \ldots, v_n\}$ and connections $\mathcal{W} = \{w_1, \ldots, w_n\} \subset \mathbb{R}^n$ be fixed beforehand. Let to each node a deterministic label $y_i \in \{-1,1\}$ be associated. The random mechanism becomes apparent through the choice of which nodes are seen during the training phase. Let the random variable $\mathcal{S}_i$ for an $i = 1, \ldots, n$ denote a randomly chosen node, governed by a fixed but unknown distribution function F_S. Then we study sets of samples $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_{n_y}\}$ which do not contain a node twice or more *(sampled without replacement)*. The algorithm observes the graph $\mathcal{G}$ and the labels of a subset of nodes y_S defined as $\mathcal{Y}_S = \{y_{\mathcal{S}_j}\}_{j=1}^{n_y}$ where $\mathcal{S}_j \subset \{1, \ldots, n\}$ and $n_y < n$. We observe a specific set $\mathcal{S}^0 \in \mathcal{S}$ with corresponding sample $\mathcal{Y}_{\mathcal{S}^o} = \{y_{\mathcal{S}_1^o}, \ldots, y_{\mathcal{S}_{n_y}^o}\}$. Now, a crucial

assumption is that this sample $\mathcal{S}^o$ is taken i.i.d. from the nodes. Formally, let the actual classification risk be defined as

$$\mathcal{R}(F_S; q) = \frac{1}{n_y} \int \sum_{i \in s} I(y_i w_i^T q < 0)\, dF_S(s), \tag{4}$$

where the indicator function $I(z < 0)$ is defined as one if $z < 0$ and zero otherwise. The empirical counterpart becomes

$$\hat{\mathcal{R}}(\mathcal{S}^o; q) = \frac{1}{n_y} \sum_{i \in \mathcal{S}^o} I(y_i w_i^T q < 0). \tag{5}$$

Application of Serfling's tail inequality as in [9, 8] gives the following result.

Theorem 1 (Complexity of Learning) *With probability* $0 < 1 - \delta < 1$*, the following inequality holds for all labelings* $q \in H_\rho$*:*

$$\mathcal{R}(F_S; q) \leq \hat{\mathcal{R}}(\mathcal{S}^o; q) + \sqrt{\left(\frac{n - n_y + 1}{n}\right) \frac{\ln |H_\rho| - \ln \delta}{2 n_y}}. \tag{6}$$

Proof: Since only a finite number of hypothesis can be chosen as stated previously, the bound follows directly from application of the Union bound on Serfling's inequality, see e.g. [12], Ch. 4. ■

It becomes apparent that a good upper-bound to $|H_\rho|$ is important for the usefulness of the bound. This theorem yields the formal motivation for the following statements *the expected loss of using the rule* '$\mathrm{sign}(w^T q)$' *on* n_y *randomly sampled nodes would not be too different from the loss observed*. The learning problem computing a hypothesis $q \in H_\rho$ which corresponds maximally with the n_y provided labels becomes

$$\min_{q \in \{-1,1\}^n} \sum_{i \in \mathcal{S}^o} I(y_i q_i < 0) \quad \text{s.t.} \quad q \in H_\rho. \tag{7}$$

Since the predictor rule '$\mathrm{sign}(w^T q)$' is to be constraint to be consistent on all nodes with the labels (see definition of H_ρ in (3)), this rule can be used to predict the label of nodes which are not in $\mathcal{S}^o$. The study of an efficient algorithm to solve this learning problem, and a formal relation to approaches based on a minimal cut is the topic of a forthcoming paper.

1.2 KINGDOM CAPACITY

We now advance to a more complex way to characterize the complexity of the class H_ρ. The Kingdom capacity (K-capacity) - denoted as $\vartheta(\rho)$ - is defined as the maximal number of nodes which can chose their label at will ('king') such that remaining nodes ('vassals') can be labeled making the king nodes consistent with theirselves. Let q_s be defined as the restriction of a vector $q \in \{-1, 1\}^n$ to the set of indices $s \subset \{1, \ldots, n\}$. Let $q_{\setminus s}$ denote the set difference $q \setminus q_s$ ($q_{\setminus s}$ is also denoted as the completion of q).

Definition 1 (Kingdom Capacity) *Given a fixed graph* $\mathcal{G} = (\mathcal{V}, \mathcal{W})$*, the K-capacity* $\vartheta(\rho)$ *is defined as follows*

$$\vartheta(\rho) = \max_{s \subset \{1,\ldots,n\}} |s| \quad \textit{s.t.} \quad \forall q_s \in \{-1,1\}^{|s|},\ \exists q_{\setminus s} \in \{-1,1\}^{n-|s|} \ \Big| \ q_i (w_i^T q) \geq \rho \quad \forall i \in s. \tag{8}$$

This definition is motivated by the following bound on the cardinality of the set H_ρ.

Theorem 2 (Bound to Cardinality) *For any graph G with n nodes and a K-capacity of $\vartheta(\rho)$, the hypothesis class H_ρ contains at most a finite number of hypotheses. This number is bounded as follows The above reasoning yields*

$$|H_\rho| \leq \sum_{d=0}^{\vartheta(\rho)} \binom{n}{d} 2^d. \tag{9}$$

Proof: At first, consider a fixed graph G with n nodes and corresponding connections $\{w_{ij} \geq 0\}_{i \neq j}$. Let s be a subset of $\{1, \ldots, n\}$ of *maximal cardinality* such that for every assignment of $+1$ either -1 to the nodes q_s, one can find a completion $q_{\setminus s} \in \{-1, 1\}^{n-|s|}$ such that for the nodes in s the rule $q_i(w_i^T q) \geq \rho$ holds. Formally

$$s \subset \{1, \ldots, n\} : \ \forall q_s \in \{-1, 1\}^{|s|}, \exists q_{\setminus s} \in \{-1, 1\}^{n-|s|} : \ q_i(w_i q) \geq \rho, \ \forall i \in s. \tag{10}$$

Remark that there may be more than one such sets $s \subset \{1, \ldots, n\}$ of the same (maximal) cardinality. More precisely, there may be maximally $\binom{n}{|s|}$ such a sets. This definition implies that the graph G permits at most $\binom{n}{|s|} 2^{|s|}$ different hypotheses. Indeed, assume that $|H_\rho| > \binom{n}{|s|} 2^{|s|}$. As q can only take two different values, there is at least one subset $s' \subset \{1, \ldots, n\}$ with higher cardinality ($|s| < |s'|$) such that all $2^{|s'|}$ possible combinations $q_{s'} \in \{-1, 1\}^{|s'|}$ satisfy the self-consistency, contradicting the assumption of maximality.

By definition, the K-capacity equals the cardinality of this maximal set s, hence we denote such a set as $s_{\vartheta(\rho)}$. Since we do not restrict the maximal number of sets $s_{\vartheta(\rho)}$, this bound holds for any given graph with n nodes and a K-capacity of $\vartheta(\rho)$, yielding (9). ■

It is seen that this reasoning is similar in spirit to Sauer's lemma and extensions as discussed e.g. in [2].

2 ASSESSING THE K-CAPACITY

2.1 UPPER-BOUND TO THE K-CAPACITY: THRESHOLDING EDGES

A first method gives an intuitive bound on the K-capacity based on neglecting the non-important connections. More precisely, if a certain connection w_{ij} is larger than all other connections $\{w_{ik}\}_{k \neq j}$, (i.e. $w_{ij} \geq \sum_{k \neq j} w_{ik} - \rho$) a cut of the former cannot be corrected by any combination of latter weights. Thus a necessary condition is that no cut may occur between node v_i and node v_j, hence reducing the maximal set of free nodes by a unit.

Proposition 1 (Bound by Thresholding Edges) *Let $\tilde{\vartheta}(\rho)$ be defined as*

$$\tilde{\vartheta}(\rho) \triangleq \max_{s \subset \{1,\ldots,n\}} |s| \quad \textit{s.t.} \quad \forall i \neq j \in s : \ w_{ij} < \sum_{k \neq j} w_{ik} - \rho, \ \textit{AND} \ w_{ij} < \sum_{k \neq i} w_{kj} - \rho. \tag{11}$$

Then $\vartheta(\rho) \leq \tilde{\vartheta}(\rho) \leq n$.

Proof: We show that $\vartheta(\rho) \leq \tilde{\vartheta}(\rho)$. Suppose $w_{ij} \geq \sum_{k\neq j} w_{ik} - \rho$ and $q_i q_j = -1$ (i.e. cut between v_i and v_j), then the following inequality holds

$$q_i \left(w_{ij} q_j + \sum_{k\neq j} w_{ik} q_k \right) \leq -w_{ij} + \sum_{k\neq j} w_{ik} \leq \rho, \tag{12}$$

hence contradicting the condition of the K-capacity in (8). This implies that no cut can be made between nodes with too strong a connection. Maximizing with this (only) necessary condition gives the upper-bound. ■

To compute this quantity, construct a graph G_ρ with the same n nodes v as previously, and weights w_ρ such that $w_{ij,\rho} = w_{ij}$ if $w_{ij} \geq \sum_{k\neq j} w_{ik} - \rho$ or $w_{ij} \geq \sum_{k\neq i} w_{kj} - \rho$, for all $i \neq j$, or zero otherwise. Then $\tilde{\vartheta}(\rho)$ simply equals the number of disconnected components in G_ρ. Remark that this notion is related to the classical notion of capacity of a graph defined as follows (see e.g. [7])

$$\text{cap}(G) = \max_{s\subset\{1,\ldots,n\}} |s| \quad \text{s.t.} \quad \forall i \neq j,\ w_{s_i,s_j} = 0. \tag{13}$$

This approach of thresholding often yields however overly pessimistic estimations ($\tilde{\vartheta}(\rho) \approx n$) of the K-capacity, especially when all non-zero weights take values of similar magnitude.

2.2 UPPER-BOUND TO THE K-CAPACITY: LAZY KINGS

A thighter upper-bound can be obtained from a convex optimization problem as follows. To introduce the methodology, we encode the problem based on the analogy with a simple strategy game. Here, we set a node to a *king* if it can chose its own label freely. The remaining nodes are a *vassal*, as they are to support their king's will. This is encoded as a binary variable ξ_i for each node v_i: $\xi_i = 1$ means that v_i is a king, $\xi_i = 0$ means that v_i is a vassal. Now it is clear that we look for the maximal number of kings which can be supported by a given graph. The first formulation assumes the kings are lazy: they are happy as long as their are enough vassal neighbors which can be enslaved when needed. This means that they need not be suspicious and vassals are thus not governed by a single king. This idea implements a necessary condition for the bound to hold, resulting in an upper-bound. Making this idea formal, we obtain that the number of vassals ($\xi_i = 0$) multiplied by their connection weight should majorate the weighted number of kings in a king's vicinity, or for all $i = 1, \ldots, n$:

$$v_i \text{ king} \quad \Rightarrow \sum_{j=1}^{n} w_{ij}\xi_j \leq \sum_{j=1}^{n} w_{ij}(1-\xi_i) - \rho. \tag{14}$$

Reformulating the left hand side and forcing the constraint to be trivial when $\xi_i = 0$ yields the expressions $2\sum_{j=1}^{n} w_{ij}\xi_j \leq (2-\xi_i)\sum_{j=1}^{n} w_{ij} - \rho$ for all $i = 1, \ldots, n$. Combining the above reasoning gives the integer programming problem

$$\max_{\xi_i\in\{0,1\}} \sum_{i=1}^{n} \xi_i \quad \text{s.t.} \quad 2\sum_{j=1}^{n} w_{ij}\xi_j \leq (2-\xi_i)\sum_{j=1}^{n} w_{ij} - \xi_i\rho, \ \ \forall i. \tag{15}$$

Relaxing the integer constraints to $\xi_i \in [0,1]$ for any $i = 1, \ldots, n$ can only increase the maximum, motivating the upper-bound

$$\overline{\theta}(\rho) \ = \ \max_{\xi_i} \sum_{i=1}^{n} \xi_i \quad \text{s.t.} \quad \begin{cases} 2\sum_{j=1}^{n} w_{ij}\xi_j \leq (2-\xi_i)\sum_{j=1}^{n} w_{ij} - \xi_i\rho \\ 0 \leq \xi_i \leq 1 \end{cases} \quad \forall i. \tag{16}$$

Remark that since (15) may only take integer values, the integer part of this value may be considered as the upper-bound without loss of generality such that $\lfloor \overline{\theta}(\rho) \rfloor \geq \vartheta(\rho)$. The upper-bound is not tight as in typical situations (i.e. choice of label of the king node), a king may not win all its neighboring vassal nodes for its personal sake. At the time of labeling, competition of the vassal nodes will divide the vassal nodes over the kings. The following section uses a fixed assignment of vassals to a unique king to obtain a lower-bound.

2.3 LOWER-BOUND TO THE K-CAPACITY: SUSPICIOUS KINGS

We show how one can compute efficiently a lower-bound to the K-capacity. what is the highest number of suspicious kings who can govern their own disjoint subgraph simultaneously and independently. To formalize this problem, let again the binary variable $\xi_i \in \{0,1\}$ denote whether a node v_i is a *vassal* ($\xi_i = 0$), or a *king* ($\xi_i = 1$). Then, define for every node v_j a vector of binary variables $\beta_j = (\beta_{j1}, \ldots, \beta_{jn})^T \in \{0,1\}^n$ encoding to which king node it will be dedicated: exactly one element of β_j is to be one, the others are zero. This property may be encoded as follows

$$\sum_{j \neq i} \beta_{ij} \leq (1 - \xi_i), \quad \forall i = 1, \ldots, n. \tag{17}$$

Furthermore, a node might be a king ($\xi_i = 0$), if it governs enough neighbors to possess a superiority over the remaining nodes when they would gather against the king (hence the denominator 'suspicious').

$$v_i \text{ is a king} \Rightarrow \sum_{j=1}^{n} w_{ji}\beta_{ji} \geq \sum_{j=1}^{n} w_{ji}(1 - \beta_{ji}) + \rho. \tag{18}$$

If not, it might thrown in its lot with a neighboring king (vassal), and the above constraint becomes obsolete. This reasoning results in the following integer programming problem

$$\theta(\rho) = \max_{\substack{\beta_{ij} \in \{0,1\} \\ \xi_i \in \{0,1\}}} \sum_{i=1}^{n} \xi_i \quad \text{s.t.} \quad \begin{cases} \sum_{j \neq i} \beta_{ij} \leq (1 - \xi_i) & \forall i \\ 2\sum_{j=1}^{n} w_{ji}\beta_{ji} \geq \xi_i \left(\sum_{j=1}^{n} w_{ji} + \rho \right) & \forall i, \end{cases} \tag{19}$$

where the binary variable ξ_i determines whether the corresponding second constraint is restrictive or trivially satisfied. A suboptimal solution to this problem is still a lower-bound. This motivates the following approximative methodology: In the first step, a convex linear programming problem is solved:

$$(\hat{\beta}_{ij}, \hat{\xi}_i) = \arg\max_{\beta_{ij}, \xi_i} \sum_{i=1}^{n} \xi_i \quad \text{s.t.} \quad \begin{cases} \sum_{j \neq i} \beta_{ij} \leq (1 - \xi_i) & \forall i \\ 2\sum_{j=1}^{n} w_{ji}\beta_{ji} \geq \xi_i \left(\sum_{j=1}^{n} w_{ji} + \rho \right) & \forall i \\ \xi_i \in [0,1] & \forall i \\ \beta_{ij} \in [0,1] & \forall i, j. \end{cases} \tag{20}$$

In the second stage, the estimates $\hat{\beta}$ and $\hat{\xi}$ are rounded to the nearest integer solutions satisfying $\beta_{ij} \in \{0, 1\}$ and $\xi_i \in \{0, 1\}$ gives a suboptimal solution. The cost resulting from this suboptimal procedure associates to the rounded arguments (say $\tilde{\theta}(\rho)$) is guaranteed to be lower or equal to $\theta(\rho)$ by construction. This reasoning is summarized in the following proposition:

Proposition 2 (Lower-bound to K-Capacity)

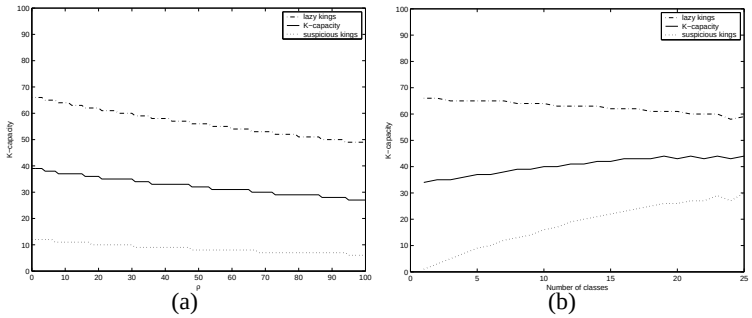
$$\vartheta(\rho) \ \geq \ \theta(\rho) \ \geq \ \tilde{\theta}(\rho). \tag{21}$$

3 ARTIFICIAL EXAMPLE

We discuss the merit of the K-capacity measure, based on a numerical case study. Consider a random graph consisting of 100 nodes, where the first fifty have a label $+1$ and the last 50 label -1. The weights are generated by the following mechanism:

$$w_{ij} \sim \begin{cases} B(p) & \text{iff } \ C(v_i) = C(v_j) \\ B(1-p) & \text{otherwise,} \end{cases} \tag{22}$$

where $B(p)$ denotes a Bernoulli distribution with parameter $0.5 < p < 1$, and $C(v_i)$ denotes the class label of node v_i. This generating mechanism is motivated as follows: nodes from the same class are likely to be strongly connected (within), while connections between the two classes are depreciated. Remark that this is another mechanism as the one discussed in the transductive setting in Subsection 2.1. The second example employs the same mechanism, but generalizes to a range of different classes.



(a) (b)

Fig. 1. *Results of an artificial example.* **(a)** *Display of the K-dimension, the lower- and upper-bound in function of the margin ρ for a random graph with 100 nodes.* **(b)** *Display of the K-dimension, the lower- and upper-bound in function of the number of different classes in the generating mechanism. It is seen that increasing both the margin ρ as well as the number of underlying different classes will make the lower- and upper-bound more tight.*

Figure 1.a gives the result of a simulation study where one graph was generated as indicated, for various parameters ρ ranging from 0 to 50, and for 100 nodes. Here we consider the case of two different classes, a parameter $p = 0.75$. The figure displays

the K-capacity as a function of the parameter ρ. The actual capacity was computed by naive (and time-consuming) enumeration, while the upper- and lower-bound respectively follow from the linear programming formulation as discussed in the previous section. Figure 1.b gives the result of a study of a set of random graph with 100 nodes, and a ranging number of classes ranging from 1 to 25. Again, both the actual K-capacity as the lower- and upper-bound discussed above are displayed as a function of the number of underlying classes.

4 DISCUSSION

We discussed a measure of capacity of a graph which characterizes the range of labelings which are self-consistent with a certain margin. Furthermore, this paper indicated how this capacity measure can be used to give probabilistic guarantees for learning in a transductive setting. In general, we described a relationship between a probabilistic approach of learning on the one hand, and combinatorial tasks such as graph cut on the other. Specifically, we indicated how the crucial concept of capacity control and regularization in learning can be mapped to graph labeling and graph cuts by using a proper extension operator. This paper is conceived as an exercise before approaching the more challenging task of proving the learnability of a MINCUT-based algorithm of the labels of *any* graph with given size. This would require an integration of the above ideas with the luckiness framework as described in [10].

References

1. Y.S. Abu-Mostafa and J.-M. St. Jacques. Information capacity of the Hopfield model. *IEEE trans. on Inf. Theory*, 31:461–464, 1985.
2. M. Anthony and P.L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.
3. A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 19–26. Morgan Kaufmann Publishers, 2001.
4. A. Blum, J. Lafferty, M.R. Rwebangaria, and R. Reddy. Semi-supervised learning using randomized mincuts. *24e International Conference on Machine Learning (ICML)*, 2004.
5. C. Cooper, M. Anthony, and G. Brightwell. The Vapnik-Chervonenkis dimension of a random graph. *Discrete Mathematics*, 138:43–56, 1995.
6. L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, 1996.
7. L. Lovasz. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979.
8. R. El-Yaniv P. Derbeko and R. Meir. Explicit learning curves for transduction and application to clustering and compression algorithms. *Journal of Artificial Intelligence Research*, 22:117–142, 2004.
9. R.J. Serfling. *Approximation Theorems of Mathematical Statistics*. John Wiley & Sons, 1980.
10. J. Shawe-Taylor, P.L. Bartlett, R.C. Williamson, and M. Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940., 1998.
11. J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE transactions on Pattern Recognition and Machine Intelligence*, 22(8), aug. 2000.
12. V.N. Vapnik. *Statistical Learning Theory*. Wiley and Sons, 1998.

Wrapper Induction: Learning (k,l)-Contextual Tree Languages Directly as Unranked Tree Automata

Stefan Raeymaekers and Maurice Bruynooghe

K.U.Leuven, Dept. of Computer Science, Celestijnenlaan 200A, 3001 Leuven, Belgium
{stefanr,maurice}@cs.kuleuven.ac.be

Abstract. A (k,l)-contextual tree language can be learned from positive examples only; such languages have been successfully used as wrappers for information extraction from web pages. This paper shows how to represent the wrapper as an unranked tree automaton and how to construct it directly from the examples instead of using the (k,l)-forks of the examples. The former speeds up the extraction, the latter speeds up the learning.

1 Introduction

The class of (k,l)-contextual tree languages is a subclass of the class of regular unranked tree languages. Similar to k-contextual string languages, it can be learned from positive examples only. It was introduced in [3] to represent wrappers that extract specific information from structured documents such as web-pages.

Here, we use unranked tree automata to represent (k,l)-contextual tree languages and describe how to learn such an automaton from the examples instead of using the (k,l)-forks of the examples. The former improves the efficiency of the extraction; the latter improves the efficiency and reduces the memory needs of the learning phase.

After some preliminaries (Section 2), (k,l)-contextual tree languages are recalled in Section 3 and automata in Section 4. Section 5 describes the construction of an automaton that accepts a fork belonging to a given set and Section 6 that of an automaton that accepts a tree belonging to a (k,l)-contextual tree language. We conclude in Section 7.

2 Preliminary Definitions

We define the alphabet Σ as a finite set of symbols. The set $T(\Sigma)$ of all finite, unranked trees with nodes labelled by elements of Σ can be recursively defined as $T(\Sigma) = \{f(w) \mid f \in \Sigma, w \in T(\Sigma)^*\}$. We usually denote $f(\epsilon)$, where ϵ is the empty sequence, by f. The subtrees of a tree are inductively defined as $sub(f(t_1,\ldots,t_n)) = \{f(t_1,\ldots,t_n)\} \cup \bigcup_i sub(t_i)$. A *tree language* is any subset of $T(\Sigma)$. The set of *(k,l)-roots* of a tree $f(t_1,\ldots,t_n)$ is the singleton $\{f\}$ if l=1; otherwise, it is the set of trees obtained by extending the root f with $(k,l-1)$-roots of k successive children of t (all children if $k > n$). Formally:

$$R_{(k,l)}(f(t_1,\ldots,t_n)) = \begin{cases} \text{if } l=1 \text{ then } \{f\} \\ \text{if } l>1 \text{ and } k>n \text{ then } f(R_{(k,l-1)}(t_1),\ldots,R_{(k,l-1)}(t_n)) \\ \text{else } \bigcup_{p=1}^{n-k+1} f(R_{(k,l-1)}(t_p),\ldots,R_{(k,l-1)}(t_{p+k-1})) \end{cases}.$$

In this formula, $f(S_1, \ldots, S_n)$, denotes the set $\{f(s_1, \ldots, s_n) \mid s_i \in S_i\}$. In a similar notational extension, $R_{(k,l)}(T)$ denotes $\bigcup_{t \in T} R_{(k,l)}(t)$, the (k,l)-roots of a set T of trees. Finally, a (k,l)-*fork* of a tree t is a (k,l)-root of any subtree of t. Thus, the set of (k,l)-forks of t can be written as $R_{(k,l)}(sub(t))$ and we denote it by $F_{(k,l)}(t)$. Then the (k,l)-forks of a set of trees T are defined as $F_{(k,l)}(T) = \bigcup_{t \in T} F_{(k,l)}(t)$.

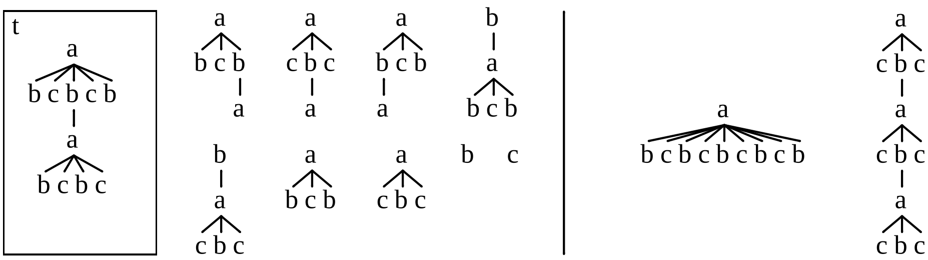
3 (k,l)-Contextual Tree Languages

3.1 Language and Learning

Definition 1. *The (k,l)-contextual tree language based on the set G of trees is defined as $L_{(k,l)}(G) = \{t \in T(\Sigma) \mid F_{(k,l)}(t) \subseteq G\}$.*

As shown in [3], the language $L_{(k,l)}(F_{(k,l)}(E))$ is the most appropriate (k,l)-contextual tree language that can be learned from a set of positive examples E as it is the most specific (k,l)-contextual language that accepts all the examples. Generalization is controlled by the choice of the parameters; they determine the minimal granularity of the building blocks (the forks from the examples) that can be used in defining the language. Negative examples can be used to adjust the parameter values [3].

Example 1. Below we show graphically the $(3,3)$-forks of a tree t. The first three of these forks are the $(3,3)$-roots of t. Two trees from the language $L_{(k,l)}(F_{(k,l)}(\{t\}))$ are shown on the right.



3.2 Wrapper Induction

To extract information from (semi)structured documents like HTML, we need to be able to select specific nodes (be it leaves or internal nodes) from a given tree. This selection is represented by marking the tree. A marked tree is an element of $T(\Sigma_X)$ with Σ a given alphabet and X a marker. The marked alphabet Σ_X is defined as $\Sigma_X = \Sigma \cup \{s_X \mid s \in \Sigma\}$. Hence the nodes of a marked tree have either a marked or an unmarked label. Marking a node s consists of replacing it by a marked equivalent s_X.

A marking of a tree is correct (with regard to the extraction task) iff the nodes to be selected are the marked ones. A marking is partially correct if every node that is marked is a node to be selected. Given a tree and a tree language that accepts partially correct markings for a given task, we can retrieve all the target nodes in a way that is linear in the number of nodes as follows: In each run, a single node of the tree is marked; if this marked tree is accepted, the marked node is a target node. Using an automaton

representation for the language, a more efficient method, that extracts all the nodes in a single run, exists ([2]).

To learn a wrapper, we need to learn a language that accepts the partially correct marked trees for the given task. The training examples consist of trees with a single node marked. Each marked node corresponds to an example of a target node of the task. As there are an infinite number of possible text nodes, all text nodes in the trees that are not recognized as a distinguishing context are generalized in a preprocessing step by replacing them with a special symbol (@) (See [3] for more details). As argued in [3], a further useful generalization is obtained by discarding all forks that do not contain a marker. As a result, typical extraction tasks require less than 5 examples.

4 String and Tree Automata

4.1 Finite String Automata

Traditionally [1], a deterministic Finite String (or Sequence) Automaton (FSA) is a tuple $\mathcal{A} = (\Sigma_i, \Sigma_o, Q_S, q_0, \delta_S, \phi_S)$ where Σ_i is a set of input symbols, called the input alphabet, Σ_o is a set of output symbols, called the output alphabet, Q_S is a finite set of states, $q_0 \in Q_S$ is the initial state, $\delta_S : (Q_S \times \Sigma_i) \rightarrow Q_S$ is a transition function from a pair consisting of a (current) state and an input symbol to a (next) state and ϕ_S is an output function $Q_S \rightarrow \Sigma_o$. We extend the transition function to a function $\hat{\delta}_S : (Q_S \times \Sigma_i^*) \rightarrow Q_S$, that is defined as $\hat{\delta}_S(q, \epsilon) = q$, and $\forall w \in \Sigma_i^*$ and $a \in \Sigma_i$, $\hat{\delta}_S(q, wa) = \delta_S(\hat{\delta}_S(q, w), a)$. The result of applying an automaton on a given sequence s of symbols from the alphabet is defined as $\phi_S(\hat{\delta}_S(q_0, s))$. Hence an FSA is a function from sequences over an input alphabet to elements of an output alphabet. Automata that have as output alphabet $\Sigma_o = \{accept, reject\}$ are called acceptors. A state for which every prefix ending in that state and all strings starting with that prefix are rejected, is called a dead state. If a state does not contain a transition for an input symbol, a transition to an implicit dead state is assumed.

4.2 Bottom-up Finite Tree Automata.

A bottom-up tree automaton works similar to an FSA. The state of a given tree is calculated by calculating the state of its children, and then getting the state from the transition function that is associated with the resulting sequence of states of the children, and the symbol in the root of the tree. The evaluation recursively descends the tree to calculate first the states of the leafs and the lower trees, hence the name bottom-up. The automaton is deterministic when the evaluation of each tree leads to a unique state. Henceforth any tree automaton mentioned will be a deterministic bottom-up tree automaton.

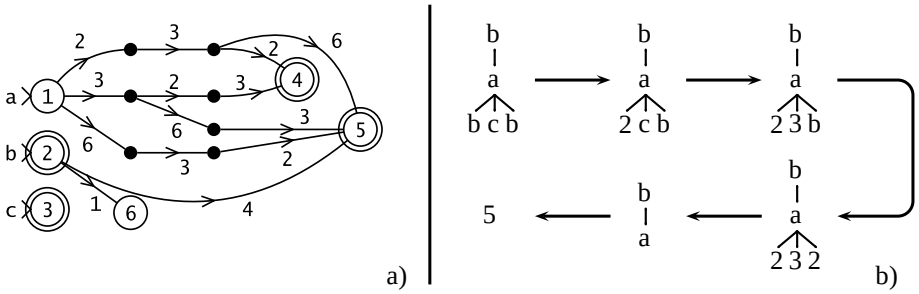
Formally we define a deterministic bottom-up Finite Tree Automaton (FTA) as a tuple $\mathcal{T} = (\Sigma_i, \Sigma_o, Q_T, \delta_T, \phi_T)$ where Σ_i is a set of input symbols, Σ_o is a set of output symbols, Q_T is a set of tree states, ϕ_T is an output function $Q_T \rightarrow \Sigma_o$ and $\delta_T : (\Sigma_i \times Q_T^*) \rightarrow Q_T$ is a transition function from a pair consisting of an input symbol and a sequence of (child) states to a (next) state, such that for each $a \in \Sigma$, the set $\{w \in Q_T^* \mid ((a, w) \rightarrow q) \in \delta_T\}$ is a regular set of strings over the alphabet Q_T.

We extend the transition function δ_T to a function $\hat{\delta}_T : (Q_T \times T(\Sigma_i)) \rightarrow Q_T$. Given a $f(s) \in T(\Sigma_i)$, $\hat{\delta}_T(f(s)) = \delta_T(f, map(\hat{\delta}_T, s))$. The function $map(func, seq)$ returns a sequence formed by the results of applying the function $func$ on each element of the sequence seq. The result of applying a FTA on a given tree t is defined as $\phi_T(\hat{\delta}_T(t))$.

4.3 Representation of the Transition Function.

The transition function of a unranked tree automaton cannot be defined by enumeration, as the number of different (a, w) pairs can be infinite. The set of state-strings leading to a given state for a given input symbol can be represented as a string automaton. We merge all these automata into a single special FSA $\mathcal{A}_\mathcal{T}$. Instead of one single initial state, a function α is used, that maps an input symbol a of the tree automaton into an initial state $\alpha(a)$ of the FSA. The outputs for this FSA are tree states, such that $\phi_S(\hat{\delta_S}(\alpha(a), w) = \delta_T(a, w)$. Note that $\mathcal{A}_\mathcal{T} = (Q_T, Q_T, Q_S, \alpha, \delta_S, \phi_S)$ (Q_T is as well input as output alphabet). When $\mathcal{T}$ is an acceptor, and ϕ_S has no value for a given element of Q_S, it is assumed to return the dead tree state. A dead tree state is a rejecting state; for every input symbol a, and every sequence w containing a dead state, it holds that $\delta_T(a, w)$ returns a dead state.

Example 2. Below is a tree automaton represented in the above formalism.



The nodes in the graph represent string states (from Q_S) of $\mathcal{A}_\mathcal{T}$. The node also shows the value of the output function. If the node is a black dot, the output is a dead state, otherwise, the number in the circle shows the tree state that is the output. A double circle indicates an accepting tree state. A transition is shown by an edge between two string states. The label is the tree state (from Q_T) that triggers the transition. This tree acceptor accepts the set of forks from Example 1. The b-part illustrates the bottom-up run of this automaton on a tree. The run ends in a state that outputs the accepting state 5, hence the given tree is accepted.

5 Fork Set Acceptor

Here, we develop an acceptor for a set of forks. A first step introduces a constructing framework for tree automata. The initial automaton is $\mathcal{T}_0 = (\Sigma_i, \Sigma_o, Q_T, \delta_T, \phi_T)$, with $\Sigma_i = \emptyset$, $\Sigma_o = \{accept, reject\}$, $Q_T = \phi_T = \emptyset$. $\mathcal{A}_\mathcal{T} = (Q_T, Q_T, Q_S, \alpha, \delta_S, \phi_S)$ with $Q_S = \alpha = \delta_S = \phi_S = \emptyset$ is an FSA used to represent δ_T. We define alternative

Algorithm 1 GetForks

Input: A tree $t = f(w)$, the values k and l

Output: An array $result$ of l elements; $result[i]$ ($i < l$) is the set of states from the (k,i)-roots of the input tree; $result[l]$ the set of states from the (k,l)-forks (as a side effect the automaton is updated).

```
1: define RecFunc(x) as GetForks(x, k, l)
2: childstates = map(RecFunc, w)
3: result[l] = ∪_{c∈childstates} c[l] % the (k,l)-forks from the subtrees of t
4: result[1] = {φ'_S(α'(f))} % the (k,1)-root of t is the only (k,1)-fork
5: if w == ε then
6:    for i: 2 .. l do result[i] = {φ'_S(α'(f))} % the (k,1)-root is the only (k,i)-root
7: else
8:    for i: 2 .. l − 1 do result[i] = ∅ % initialisation
9:    for i = l downto 2 do
10:      if length(childstates[i − 1]) < k then
11:         sequences = {childstates[i − 1]}
12:      else
13:         sequences = all sequences of k consecutive sets in childstates[i − 1]
14:      end if
15:      for all seq ∈ sequences do
16:         for every tuple tup obtained by taking one element from each set of seq do
17:            result[i] = result[i] ∪ {φ'_S(δ̂'_S(α'(f), tup))}
18:         end for
19:      end for
20:   end for
21: end if
```

functions α', δ'_S, and ϕ'_S for respectively α, δ_S, and ϕ_S. The function α' is defined such that $\alpha'(a) = \alpha(a)$ for every $a \in \Sigma_i$. For every $a \notin \Sigma_i$, the following side effects are performed: a is added to Σ_i, a new state s is added to Q_s, and α is extended with $\alpha(a) = s$; finally the state s is returned as function value. Similarly, $\delta'_S(s,t)$ returns $\delta_S(s,t)$, when the latter is defined. Otherwise a new state q is added to Q_S, δ_S is extended with $\delta_S(s,t) = q$, and q is returned as function value. Finally, $\phi'_S(s) = \phi_S(s)$ when defined, otherwise a new tree state t is added to Q_T, ϕ_T is extended with $\phi_T(t) = reject$, ϕ_S is extended with $\phi_S(s) = t$, and t is returned as function value. Below we assume that the function $\hat{\delta}_T$ is defined based on these alternative functions.

To construct a fork set acceptor, we create an initial automaton and, for every fork in the example trees of our language, we call $\hat{\delta}_T$ and change the output of the final tree state from $reject$ to $accept$. To create an acceptor for the set of marked forks of a tree, we discard forks that are not marked. As forks from the same example tree have parts in common, some redundant work is done.

To avoid the redundancy, the algorithm GetForks below processes all forks from an example tree at once. Besides updating the automaton, it outputs an array of l sets of output states. For $i < l$, the i^{th} element contains the states associated with all the (k,i)-roots of the input tree while the l^{th} element contains the states associated with all the (k,l)-forks of the input tree. Thanks to the recursive call, the states for the children

of a node are calculated only once and reused in the different combinations in which they form the (k, p)-roots of a node (for $2 \leq p \leq l$).

Note that the automaton of Example 2 is a (3,3)-fork set acceptors for the forks from the tree t of Example 1. It is the minimization of the output from our algorithm.

Marked Fork Set Acceptor The fork set acceptor can be further refined to disregard forks without a marked node. With only one node in the tree marked, only the roots from the marked node itself, and the roots from its $l - 1$ direct ancestors that contain the marker are to be processed. Hence from the children of every ancestor, only the k-1 siblings before and after the child that is also an ancestor (or the marked node itself) need to be considered. Starting from the marked node, and using a data structure where one has access to the parent, one can construct a variant of the algorithm that disregards the irrelevant parts of the input tree.

6 (k, l)-Contextual Tree Acceptor

To check whether a tree belongs to a (k, l)-contextual tree language defined by a set of forks, we need to run the associated fork set acceptor on each of the forks of that tree (and stop as soon as one is rejected). As different forks overlap, such an acceptor will perform a lot of redundant work. Instead, we use the fork set acceptor to construct a tree automaton that can process any tree bottom-up. To do so, we need to encode in the automaton the transitions the fork set acceptor can do in all possible runs and to organize it in such a way that it can simulate the fork acceptor runs for the forks in a given tree. Note that the number of possible runs of a fork set acceptor is finite as the set of (k, l)-forks is finite. Below, we sketch the construction of such an automaton.

Note that our (k, l)-contextual tree acceptor will associate an accepting state with each node of an accepted tree. Indeed, if a tree is accepted, also its subtrees are accepted as the forks of a subtree are a subset of the forks of the original tree. Hence, every tree state of the (k, l)-contextual tree acceptor will be either an accepting state or the dead tree state.

6.1 Construction

The state associated with a tree node by the (k, l)-contextual tree acceptor corresponds to different states in different runs of the fork set acceptor because the node under consideration belongs to different forks. Therefore, during construction, we have to maintain "internal" information about the state of the fork set acceptor in each of the possible runs. The internal representation of the string states (used to represent the transition function of the automaton under construction) is a two dimensional array. Elements in column i contain states of nodes that are the i^{th} child of its parent in the runs of the fork set acceptor represented by the element, while elements in row j contain states of nodes that are j levels deep in the runs of the fork set acceptor represented by the element. In fact, one can do with $k-1$ rows (if extending the internal representation with a control bit that indicates whether one has already had k children in the sequence) and with $l-1$ columns, because the root need not be considered as a child. The internal

representation of the tree states is a one dimensional array. The position corresponds to the level of the node in the represented runs of the fork set acceptor, i.e., similar to output of GetForks, the first element contains the states of the forks of which the node is root, while the other rows contain states of the node when it is a (k, p)-root ($p < l$); info to be used for computing the states of its parent node.

A high level description of the algorithm is shown in Algorithm 2. In what follows, we add a superscript F when referring to functions of the fork set acceptor. The main loop processes elements from an agenda. It contains string states for which the possible transitions (represented by the fork set acceptor) have yet to be computed. This agenda is initialised with a state s for every symbol f in Σ_i. The internal representation of such a state s is the empty matrix (and a false control bit). The tree state that is the output of s is a dead state when, $\phi_S^F(\alpha^F(f))$ is a rejecting state. Otherwise, ϕ_S is extended with $\phi_S(s) = t$ with t a new tree state . Its internal representation is a one element array containing the singleton $\phi_S^F(\alpha^F(f))$ (it is not yet inserted in Q_T).

Algorithm 2 Create (k, l)-contextual tree acceptor

Input: A (k, l)-forks set acceptor $\mathcal{T}_F$ and an empty (k, l)-contextual tree acceptor
Output: The (k, l)-contextual tree acceptor $\mathcal{T} = (\Sigma_i, \Sigma_o, Q_T, \mathcal{A}_T, \phi_T)$.

1: $visited = \emptyset$; initialize $agenda$
2: **while** $agenda \neq \emptyset$ **do**
3: $current =$ get and remove string state from agenda
4: **if** $\phi_S(current) \notin Q_T$ **then**
5: add $\phi_S(current)$ to Q_T.
6: **for** all $s \in visited$ **do**
7: $q =$ result of transition from s, given $\phi_S(current)$ (simulated by $\mathcal{T}_F$).
8: **if** $q \notin (agenda \cup visited)$ **then**
9: extend $\mathcal{A}_T$ with $\delta_S(s, \phi_S(current)) = q$
10: add q to $agenda$
11: **end if**
12: **end for**
13: **end if**
14: **for** all $t \in Q_T$ **do**
15: $q =$ result of transition from $current$, given t (simulated by $\mathcal{T}_F$).
16: **if** $q \notin (agenda \cup visited)$ **then**
17: extend $\mathcal{A}_T$ with $\delta_S(current, t) = q$
18: add q to $agenda$
19: **end if**
20: **end for**
21: add $current$ to $visited$
22: **end while**

In processing a string state $current$ from the agenda. It is first checked whether t, the tree state it outputs is already in Q_T. If not, it is added and, for each already processed string state s (in $visited$), the fork set acceptor is used to find (and eventually

create) the string state q that is the result of the transition triggered by t (step 7)[1]. If q is new, it is added to the agenda and the transition function is extended. Finally, for each tree state t, the fork set acceptor is used to find the string state q that is the result of the transition from $current$ that is triggered by t. Such q are processed as above.

6.2 (k, l)-Contextual Marked Tree Acceptor

To obtain a marked tree set acceptor, we want to start from an acceptor that accepts a set of marked forks but also any unmarked fork. For the latter, more simple is to create an acceptor that accepts any unmarked tree. Then one can create an acceptor that is the union of the latter with the marked fork acceptor. Given this acceptor, one can proceed as above to obtain a marked tree acceptor. The latter accepts all trees with a marked node, for which the set of marked forks is a subset of a given set of marked forks.

7 Conclusion

We proposed in this paper to represent (k, l)-contextual tree languages [3] by unranked bottom-up tree automata. We introduced a method to learn the (k, l)-contextual tree acceptor directly from a given set of examples and discussed the adaptation for use in the wrapper induction task as it is defined in [3]. In a first step we defined an algorithm that, given a set of example trees, constructs an acceptor for the (k, l)-forks of these trees. In a second step this fork set acceptor is used to generate the (k, l)-contextual tree acceptor for the same parameters k and l.

The direct construction from examples speeds up the learning phase while the use of the automata representation reduces the memory needs during learning and extraction. The new approach facilitates a more efficient extraction (see [2], showing speed-up factors between 10 and 50) in the application of wrappers for information extraction from structured documents. Moreover, one can extract multiple fields within a single automaton, by combining their acceptors. The parameters k and l can be different for each field. This results in further speed-ups.

References

1. E.F. Moore. Gedanken-experiments on sequential machines. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.
2. Stefan Raeymaekers and Maurice Bruynooghe. Extracting information from structured documents with automata in a single run. In *Proc. 2nd Int. Workshop on Mining Graphs, Trees and Sequences (MGTS 2004, Pisa, Italy)*, pages 71–82, Pisa, Italy, 2004. University of Pisa.
3. Stefan Raeymaekers, Maurice Bruynooghe, and Jan Van den Bussche. Learning (k, l)-contextual tree languages for information extraction. In *ECML*, pages 305–316, 2005.

[1] The internal representations are used to identify the different transitions made by the fork set acceptor. We omit the details.

Mining Discriminative Patterns from Graph Structured Data with Constrained Search

Kiyoto Takabayashi[1], Phu Chien Nguyen[1], Kouzou Ohara[1], Hiroshi Motoda[2], and Takashi Washio[1]

[1] I.S.I.R., Osaka University,
8-1, Mihogaoka, Ibaraki, Osaka, 567-0047, Japan
{kiyoto_ra,chien,ohara,washio}@ar.sanken.osaka-u.ac.jp
[2] AFOSR/AOARD
7-23-17, Roppongi, Minato-ku, Tokyo 106-0032, Japan
hiroshi.motoda@aoard.af.mil

Abstract. A graph mining method, Chunkingless Graph-Based Induction (Cl-GBI), finds typical patterns that appear in graph structured data by the operation called chunkingless pairwise expansion which generates pseudo-nodes from selected pairs of nodes in the data. Cl-GBI enables to extract overlapping subgraphs, while it requires more time and space complexities. Thus, it happens that Cl-GBI cannot extract patterns that need be large enough to describe characteristics of data within a limited time and a given computational resource. In such a case, extracted patterns may not be so much of interest for domain experts. To mine more discriminative patterns which cannot be extracted by the current Cl-GBI, we introduce a search algorithm guided by domain knowledge or interests of domain experts. We further experimentally show that the proposed method can efficiently extract more discriminative patterns using both synthetic and real world datasets.

1 Introduction

Over the last decade, there has been much research work on data mining which intends to find useful and interesting knowledge from massive data. A number of studies have been made in recent years especially on mining frequent patterns from graph structured data, or simply graph mining because of the high expressive power of graph representation [1, 13, 6, 12, 4, 5].

Chunkingless Graph Based Induction (Cl-GBI) [8] is an extension of Graph Based Induction (GBI) [13] that can extract typical patterns from graph structured data by stepwise pair expansion, i.e., by recursively chunking two adjoining nodes. Similarly to GBI, Cl-GBI adopts the stepwise pair expansion principle, but never chunks adjoining nodes and contracts the graph. Instead, Cl-GBI regards a pair of nodes as a *pseudo node* and assigns a new label to it. This operation can fully solve the reported problems caused by chunking, i.e., ambiguity in selecting nodes to chunk and incompleteness of the search. However Cl-GBI requires more time and space compelexities to gain the ability of extracting overlapping patterns. Thus, it happens that Cl-GBI cannot extract patterns

that need be large enough to describe characteristics of data within time and space limitation. In such a case, extracted patterns may not be so much of interest for domain experts.

To improve the search efficiency, in this paper, we propose a method of guiding the search of Cl-GBI using domain knowledge or interests of domain experts. The basic idea is adopting patterns representing knowledge or interests of domain experts as constraints on the search, in order to effectively restrict the search space and extract more discriminative or interesting patterns than those which can be extracted by the current Cl-GBI. We also experimentally show the effectiveness of the proposed search method by applying the constrained Cl-GBI to a synthetic dataset and the hepatitis dataset which is a real world dataset.

In this paper, we deal with only connected labeled graphs, and use information gain [10] as the discriminativity criterion. In what follows, "a pair" denotes a pair of adjoining nodes in a graph.

2 Constrained Search for Cl-GBI

2.1 Chunkingless Graph-Based Induction(Cl-GBI)

Stepwise pair expansion is an essential operation in GBI, which recursively generates new nodes from pairs of two adjoining nodes selected according to a certain criterion based on frequency, and replaces all of its occurrences in graphs with a node having a newly assigned label. Namely each graph is rewritten each time a pair is chunked, and never restored in any subsequent chunking [3]. Although thanks to this chunking mechanism, GBI can efficiently extract patterns from either a huge single graph or a set of graphs, it involves ambiguity in selecting nodes to chunk, which causes a crucial problem, i.e., possibility of overlooking some overlapping subgraphs due to inappropriate chunking order. Beam search adopted by Beam-wise GBI(B-GBI) [6] can alleviate this problem by chunking the b (beam width) most frequent pairs and copying each graph into respective states, but not completely solve it because chunking process is still involved.

In contrast to GBI and B-GBI, Cl-GBI does not chunk a selected pair, but regards it as a *pseudo node* and assigns a new label to it. Thus, graphs are not "compressed" nor copied over the iterative *pseudo-chunking* process. We refer to each iteration in Cl-GBI as "level". The algorithm of Cl-GBI is shown in Fig. 1. The search of Cl-GBI is controlled by the following parameters: a beam width b, the maximal number of levels of pseudo-chunking N, and a frequency threshold θ. In other words, at each level, the b most frequent pairs are selected from a set of pairs whose frequencies are not less than θ, and are pseudo-chunked.

2.2 Patterns Used as Constraints

The current Cl-GBI blindly extracts a huge number of frequent pairs without any clues other than frequency. However, if the goal is finding patterns which are

[3] This does not mean that the link information of the original graphs is lost. It is always possible to restore how each node is connected in the extracted subgraphs.

Input. A graph database D, a beam width b, the maximal number of levels of pseudo-chunking N, a frequency threshold θ
Output. A set of typical patterns S
Step 1. Extract all the pairs consisting of two connected nodes in the graphs, register their positions using node id (identifier) sets. From the 2nd level on, extract all the pairs consisting of two connected nodes with at least one node being a new pseudo-node.
Step 2. Count frequencies of extracted pairs and eliminate pairs whose frequencies count below θ.
Step 3. Select the b most frequent pairs from among the remaining pairs at Step 2 (from the 2nd level on, from among the unselected pairs in the previous levels and the newly extracted pairs). Each of the b selected pairs is registered as a new node. If either or both nodes of the selected pair are not original but pseudo-nodes, they are restored to the original patterns before registration.
Step 4. Assign a new label to each pair selected at Step 3 but do not rewrite the graphs. Go back to Step 1.

Fig. 1. Algorithm of Cl-GBI

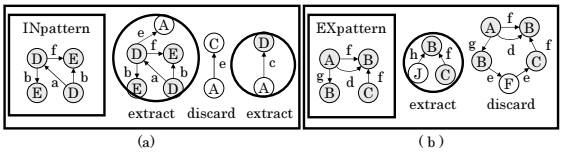


Fig. 2. Examples of INpatterns and EXpatterns

either discriminative or of interest for domain experts, the current method is too naive and inefficient in both time and space complexities. For example, when we analyzed the hepatitis dataset [11] provided by Chiba University Hospital with Cl-GBI, domain experts (medical doctors) expected that patterns interesting for them were extracted, but in fact we could not find satisfactory ones.

Therefore, in this paper, we introduce domain knowledge or interests of domain experts and impose them as constraints on patterns extracted in Cl-GBI. We represent such domain knowledge and interests as graphs and call them the *constraint patterns*. Although various types of constraint patterns can be considered, in this paper, we focus on the following two types of patterns: patterns to be included in extracted patterns and patterns not to be included in them. We refer to them as *INpatterns* and *EXpatterns*, respectively, and define the following two types of constraints: "*extracted patterns must include INpatterns*" (Constraint 1) and "*extracted patterns must not include EXpatterns*" (Constraint 2).

Figures 2 (a) and (b) show the examples of Constraints 1 and 2, respectively. Note that in case of Constraint 1, not only patterns including the given INpatterns, but also patterns including at least one of their proper subgraphs should be extracted in order not to prevent patterns satisfying the imposed constraints from being generated in the succeeding steps based on the stepwise pair expansion principle. The case is illustrated at the far right in Fig. 2 (a). In addition, in case of Constraint 1, we can discard a pair if it does not include any node/link labels appearing in the given INpatterns as shown in Fig. 2 (a) because such a pair can never grow to a pattern that includes at least one of the given INpatterns.

2.3 Design of Constrained Search

To guide the search process of Cl-GBI using INpatterns/EXpatterns, we have to check if an enumerated pair includes them, which requires additional subgraph isomorphism checking known to be NP-complete [2]. Therefore, to reduce the computational cost, we should detect pairs that have no possibility of including constraint patterns before the checking. For that detection, we define two conditions based on the number of node/link labels in a pattern.

First of all, for two arbitrary pairs, or patterns x and y, we define a quantity T_{num} as follows:

$$T_{num}(x, y) = \sum_{L_k \in L(y)} f(x, L_k), \tag{1}$$

where $L(y)$ is a set of labels appearing in y, and $f(x, L_k)$ is the number of occurrences of the label $L_k \in L(y)$ in x. Note that if x is identical to y, $T_{num}(x, y)$ must be equal to $T_{num}(y, y)$. Similarly, $T_{num}(x, y)$ must be greater than $T_{num}(y, y)$ if y is a subgraph of x. Consequently, given a constraint pattern T_j, we can skip subgraph isomorphism checking for an enumerated pattern P_i if $T_{num}(P_i, T_j) < T_{num}(T_j, T_j)$ because P_i never includes T_j.

Furthermore, it is noted that in order for P_i to include T_j, for every label appearing in T_j, the number of its occurrences in P_i has to be greater than or equal to that in T_j. Namely, we can skip subgraph isomorphism checking for P_i if P_i does not satisfy this condition. To check this condition, we define the following boolean value P_{info} for two patterns x and y.

$$P_{info}(x, y) = \bigwedge_{L_k \in L(y)} p(x, y, L_k), \tag{2}$$

where

$$p(x, y, L_k) = \begin{cases} true & \text{if } f(x, L_k) \geq f(y, L_k), \\ false & \text{otherwise.} \end{cases}$$

If $P_{info}(P_i, T_j)$ is $true$, then subgraph isomorphism checking has to be done; otherwise it can be skipped.

These ideas are summarized in Fig. 3 as the algorithm, which is invoked in the algorithm shown in Fig. 1 and provides a set of candidate pairs to be pseudo-chunked at each level. $PD(P_i, T_j)$ in Fig. 3 is the procedure for subgraph isomorphism checking, which returns true if P_i includes T_j; otherwise false.

3 Experimental Evaluation

To evaluate the proposed method, we implemented Cl-GBI with the algorithm shown in Fig. 3 on PC (CPU: Pentium 4 3.2GHz, Memory: 4GB, OS: Fedora Core release 3) in C++, and applied this *constrained Cl-GBI* to both synthetic and real-world datasets consisting of directed graphs. The current system has a limitation that either INpattern constraints or EXpattern constraints can be

```
ExtPair(D, T, L, L_v, M)
Input: a database D, a set of constraint patterns T, the current level L_v,
       a set of extracted pairs L (initially empty),
       the constraint mode M (either "INpattern" or "EXpattern");
Output: a set of extracted pairs L with newly extracted pairs;
begin
  if L_v = 1 then
    if M = "INpattern" then
      Enumerate pairs in D, which consist of nodes or links
      appearing in T, and store them in E;
    else
      Enumerate all the pairs in D and store them in E;
  else
    Enumerate pairs, which consist of one or both
    pseudo nodes in L, and store them in E;
  for each P_i ∈ E begin
    if P_i is marked then L := L ∪ {P_i}; next;
    else register := 1;
    for each T_j ∈ T begin
      if T_num(P_i, T_j) ≥ T_num(T_j, T_j) then
        if P_info(P_i, T_j) = true then
          if M = "INpattern" then
            if PD(P_i, T_j) = true then mark P_i;
          else
            if PD(P_i, T_j) = true then
              discard P_i; register := 0; break;
    end
    if register = 1 then L := L ∪ {P_i};
  end
  return L;
end
```

Fig. 3. Algorithm of the constrained pattern extraction

imposed at a time. We verified the efficiency of the proposed method with the synthetic dataset, and also confirmed that it could extract patterns which are more discriminative than those by the current Cl-GBI with the real-world one.

3.1 Synthetic Dataset

Experimental Settings: The synthetic dataset was generated in a random manner same as [9], and divided into two classes of equal size, "active" and "inactive". Then, as discriminative patterns, we generated 4 kinds of subgraphs, or "basic patterns" shown in Fig. 4, and embedded them in transactions of the class "active". In Fig. 4, "f" and "IG" denote frequency and information gain, respectively. The statistics on the size of resulting graphs are shown in Table 1.

In this experiment, we used as INpatterns a subgraph of each basic pattern shown in Fig. 4 separately to extract the corresponding basic pattern, and set the parameters of Cl-GBI as follows: $b = 5, 7, 10$,$N = 10$, and $\theta = 0\%$.

Experimental Results: We observed the computation time and the level at which the basic pattern was extracted by the constrained Cl-GBI in each case. The results are summarized in Table 2, in which t and level denote the observed computation time and level, respectively, and the values in the parentheses are

Table 1. Size of graphs of the synthetic dataset and the hepatitis dataset

class	Synthetic		Hepatitis	
	active	inactive	R	N
number of graphs	150	150	38	56
average number of nodes in a graph	50	50	104	112
total number of nodes	7,524	7,502	3,944	6,296
kinds of node labels	25		12	
average number of links in a graph	498	495	108	117
total number of links	74,631	74,198	4,090	6,577
kinds of link labels	25		30	

Table 2. Experimental results for the synthetic dataset

constraint	$b = 5$			$b = 7$			$b = 10$		
pattern	t[sec]	level	IG	t[sec]	level	IG	t[sec]	level	IG
a2	119(5976)	4(10)	0.0421	133(6537)	4(8)	0.0421	170(6810)	4(5)	0.0603
b2	42(–)	4(–)	0.0421	32(–)	3(–)	0.0421	42(–)	3(–)	0.0603
c2	168(–)	6(–)	0.0421	166(–)	5(–)	0.0421	150(–)	4(–)	0.0603
d2	167(–)	6(–)	0.0421	92(–)	4(–)	0.0421	75(–)	3(–)	0.0603

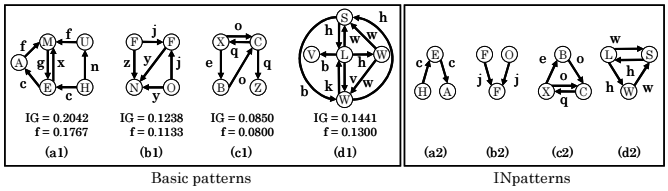


Fig. 4. Basic patterns and INpatterns used in the experiments for the synthetic dataset

corresponding results by the current Cl-GBI. "–" represents that the current Cl-GBI could not extract the basic pattern. The column "IG" in Table 2 denotes the maximal information gain achieved by the current Cl-GBI at the same level with the same parameter settings.

From this table, we can see that the constrained Cl-GBI succeeded in extracting the basic pattern in all the cases at an early stage, while the current Cl-GBI extracted only patterns having considerably less information gain and could not extract some of basic patterns. From these results, it is said that the constrained Cl-GBI can extract patterns which are more discriminative than those by the current Cl-GBI in a less computation time and smaller computational resources.

3.2 Real-world Dataset

Experimental Settings: As the real-world dataset, we used the two classes in the hepatitis dataset, *Response* and *Non-Response*, denoted by R and N, respectively [3]. R consists of patients to whom the interferon therapy was effective, while N consists of those to whom it was not effective. We converted the records of each patient into a graph in the same way as [3]. The statistics on the size of resulting graphs are shown in Table 1.

In this experiment, we used 4 sets of INpatterns shown in Fig. 5, in which (a) to (c) represent typical examination results for patients belonging to R [7],

Table 3. Experimental results for the hepatitis dataset

	time[sec]	max information gain
original	44,973	0.1139 (L:4, t:1292)
No.1	9,355	0.1076 (L:3, t:18)
No.2	6,893	0.1698 (L:5, t:376)
No.3	20,495	0.1110 (L:3, t:55)
No.4	4,970	0.1297 (L:4, t:39)

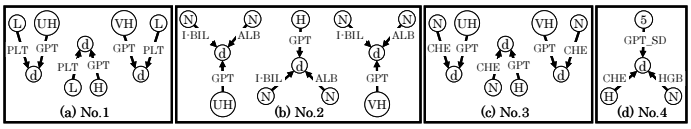


Fig. 5. INpatterns used in the experiments for the hepatitis dataset

while (d) is the most discriminative pattern extracted by the current Cl-GBI. We refer to the pattern which is the most discriminative one among extracted patterns as the *MDpattern*. The node with the label "d" in Fig. 5 represents a certain point of time. For example, the leftmost pattern in Fig. 5 (a) means that at a certain point of time, the value of GPT (glutamic-pyruvic transaminase) is High and the value of PLT (platelet) is Low. The parameters of Cl-GBI were set as follows: $b = 10$, $N = 10$, and $\theta = 0\%$.

Experimental Results: We observed the computation time and information gain of the MDpattern in each case. The results are shown in Table 3, in which the row "original" contains the results by the current Cl-GBI with the same parameter settings. Namely, the MDpattern shown in Fig. 5 (d) is identical to that in the case of "original". "L" and "t" in parentheses denote the level and time[sec] spent to extract the MDpattern, respectively.

From Table 3, it is found that the MDpatterns extracted by the constrained Cl-GBI are more discriminative than the MDpattern by the current Cl-GBI in the cases of No.2 and No.4. In addition, the computation times in all the 4 cases using INpatterns are much less than in the case of the current Cl-GBI. From these results, we can say that given appropriate constraints, the constrained Cl-GBI could efficiently extract patterns which are more discriminative than those by the current Cl-GBI. In addition, note that the INpattern used in the case of No.4 which is the MDpattern obtained by the current Cl-GBI works as a good constraint. From this result, it is expected that running the constrained Cl-GBI repeatedly with a small L using the MDpattern extracted by the previous run as the new INpattern might allow us to extract discriminative patterns in a less computation time. Verifying this expectation is one of our future work.

4 Conclusion

In this paper, we proposed a constrained search method that effectively restricts the search space of Cl-GBI by imposing domain knowledge or interests of domain experts as constraints on patterns to be searched, and embedded it in Cl-GBI,

resulting in the constrained Cl-GBI. Experimental results showed that given appropriate constraints, the constrained Cl-GBI can extract more discriminative patterns in a less computation time than the current Cl-GBI. In addition, the results also showed the possibility that discriminative patterns extracted in earlier steps in the search may work as good constraints in the constrained Cl-GBI.

As future work, we plan to further evaluate the constrained Cl-GBI by comparing it with other graph mining methods including ones based on Inductive Logic Programming, and to verify the resulting patterns cooperating with domain experts such as medical doctors.

References

1. Cook, D. J. and Holder, L. B.: Substructure Discovery Using Minimum Description Length and Background Knowledge. Artificial Intelligence Research, Vol. 1, pp. 231–255, (1994).
2. Fortin, S.: The Graph Isomorphism Problem. Technical Report TR96-20, Department of Computer Science, University of Alberta, (1996).
3. Geamsakul, W., Yoshida, T., Ohara, K., Motoda, H., Yokoi, H., and Takabayashi, K.: Constructing a Decision Tree for Graph-Structured Data and its Applications. Fundamenta Informaticae Vol. 66, No.1-2, pp. 131–160, (2005).
4. Inokuchi, A., Washio, T., and Motoda, H.: Complete Mining of Frequent Patterns from Graphs: Mining Graph Data. Machine Learning, Vol. 50, No. 3, pp. 321–354, (2003).
5. Kuramochi, M. and Karypis, G.: An Efficient Algorithm for Discovering Frequent Subgraphs. IEEE Trans. Knowledge and Data Engineering, Vol. 16, No. 9, pp. 1038–1051, (2004).
6. Matsuda, T., Motoda, H., Yoshida, T., and Washio, T.: Mining Patterns from Structured Data by Beam-wise Graph-Based Induction. Proc. of DS 2002, pp. 422–429, (2002).
7. Motoyama, S., Ichise, R., and Numao, M.: Knowledge Discovery from Inconstant Time Series Data. JSAI Technical Report, SIG-KBS-A405, pp. 27–32, in Japanese, (2005).
8. Nguyen, P. C., Ohara, K., Motoda, H., and Washio, T.: Cl-GBI: A Novel Approach for Extracting Typical Patterns from Graph-Structured Data. Proc. of PAKDD 2005, pp. 639–649, (2005).
9. Nguyen, P. C., Ohara, K., Mogi, A., Motoda, H., and Washio, T.: Constructing Decision Trees for Graph-Structured Data by Chunkingless Graph-Based Induction. Proc. of PAKDD 2006, pp. 390–399, (2006).
10. Quinlan, J. R.: Induction of decision trees. Machine Learning, Vol. 1, pp. 81–106, (1986).
11. Sato, Y., Hatazawa, M., Ohsaki, M., Yokoi, H., and Yamaguchi, T.: A Rule Discovery Support System in Chronic Hepatitis Datasets. First International Conference on Global Research and Education (Inter Academia 2002), pp. 140–143, (2002).
12. Yan, X. and Han, J.: gSpan: Graph-Based Structure Pattern Mining. Proc. of ICDM 2002, pp. 721–724, (2002).
13. Yoshida, K. and Motoda, H.: CLIP: Concept Learning from Inference Patterns. Artificial Intelligence, Vol. 75, No. 1, pp. 63–92, (1995).

Two connectionist models for graph processing: an experimental comparison on relational data

Werner Uwents[1], Gabriele Monfardini[2]
Hendrik Blockeel[1], Franco Scarselli[2], and Marco Gori[2]

[1] Department of Computer Science, Katholieke Universiteit, Leuven, Belgium
{hendrik,werner}@cs.kuleuven.be
[2] Dipartimento di Ingegneria dell'informazione
Università di Siena, Siena, Italy
{marco,monfardini,franco}@dii.unisi.it

Abstract. In this paper, two recently developed connectionist models for learning from relational or graph-structured data, i.e. Relational Neural Networks (RelNNs) and Graph Neural Networks (GNNs), are compared. We first introduce a general paradigm for connectionist learning from graphs that covers both approaches, and situate the approaches in this general paradigm. This gives a first view on how they relate to each other. As RelNNs have been developed with learning aggregate functions in mind, we compare them to GNNs for this specific task. Next, we compare both with other relational learners on a number of benchmark problems (mutagenesis, biodegradability). The results are promising and suggest that RelNNs and GNNs can be a viable approach for learning on relational data. They also point out a number of differences in behavior between both approaches that deserve further study.

1 Introduction

Object localization [1], image classification [2], natural language processing, bioinformatics, web page scoring, social networks analysis and relational learning are examples of applications where the information of interest is encoded into the relationships between a set of basic entities. In those domains, data are suitably represented as sequences, trees, and, more generally, directed or undirected graphs. In fact, nodes can be naturally used to denote concepts with edges specifying their relationships.

For example, relational databases contain information that is naturally represented by graphs: each tuple of a relation can be denoted by a node, while the relationships between different tuples are represented by edges. The nodes of the graph have labels that correspond to the attributes of the tuples, while some edge labels may define the direction of the edge if the relationships between tuples are not symmetric. The application of machine learning techniques to relational databases is receiving an increasing interest from researchers. A common application consists of predicting an attribute of a relation, i.e. learning from examples a function $\varphi(\boldsymbol{G}, n)$ that takes as input a database $\boldsymbol{G}$ and a tuple n and returns an attribute of n. When we consider the graphical representation, φ is a function on graphs, $\boldsymbol{G}$ is the graph representing the database and n is a node.

Recently, new connectionist models were proposed, that are capable to elaborate graphs and trees directly, embedding the relationship between nodes into the processing scheme. They extend support vector machines [3, 4], neural networks [5–7] and SOMs [8] to structured data. The main idea underlying these methods is to automatically obtain a flat internal representation of the information collected in the graphs. While in kernels for graphs the internal encoding is defined by the selected kernel, in neural network models it is automatically learned from examples. SOMs–SD are similar, but implement an unsupervised learning framework, instead of a supervised one.

In this paper, we discuss and evaluate two connectionist models, i.e. Relational Neural Networks (RelNNs) [9, 10] and Graph Neural Networks (GNNs) [7], that have been proposed recently. The models will be described in the next section, introducing a general paradigm that covers most of the existing neural network approaches to graph processing. Moreover, an experimental evaluation is carried out in section 4, to devise some differences between the two models and to assess their computational capabilities. Finally, the conclusions are drawn in Section 5.

2 Terminology

In the following explanation, a *graph* $\boldsymbol{G}$ is a pair ($\boldsymbol{N}$,$\boldsymbol{E}$), where $\boldsymbol{N}$ is a set of *nodes* (or vertices), and $\boldsymbol{E}$ is a set of *edges* (or arcs) between nodes: $\boldsymbol{E} \subseteq \{(u, v)|u, v \in \boldsymbol{N}\}$. We assume that edges are *directed*, i.e. each edge (u, v) is ordered and it has a head v and a tail u. The children ch$[n]$ and the parents pa$[n]$ of a node n are defined by ch$[n] = \{u|\,(n, u) \in \boldsymbol{E}\}$ and pa$[n] = \{u|\,(u, n) \in \boldsymbol{E}\}$. Nodes and edges can be labeled, and labels are represented by $\boldsymbol{l}_n$ and $\boldsymbol{l}_{(u,v)}$, respectively, while $\boldsymbol{l}$ defines all the labels of the graph. Each node can also be labeled with a type k_n. The type k_n belongs to a finite set $\boldsymbol{K}$ and defines which object is represented by the node or, in other words, the table the tuple belongs to. The operator $|\cdot|$ denotes the cardinality or the absolute value of its argument, according to whether it is applied to a set or to a number. The goal is to approximate a target function $\tau(\boldsymbol{G}, n) \in \mathbb{R}^m$ that maps a graph $\boldsymbol{G}$ and one of its nodes n into a vector of real numbers. There can be one target for a whole graph (graph classification), or a target for each node (node classification).

3 A general paradigm

Although there is a difference in motivation between graph and relational neural networks [7,9], which is not discussed in this paper, they can be described using a common processing paradigm. In this section, they will both be situated in this general framework. The main idea is to attach to each node n a vector of real numbers $\boldsymbol{x}_n \in \mathbb{R}^s$ called *state*. The state contains a description of the concept represented by the node n and is evaluated locally at each node, i.e. the processing scheme is distributed. The goal here is to combine label information with subsymbolic contributions embedded in graph topology. Such a task is accomplished computing $\boldsymbol{x}_n$ as the output of a parametric function $f_{\boldsymbol{w}}^{k_n}$, called *state transition function*, that combines the information attached to node n and to its children ch$[n]$

$$\boldsymbol{x}_n = f_{\boldsymbol{w}}^{k_n}(\boldsymbol{l}_n, \boldsymbol{x}_{\mathsf{ch}[n]}, \boldsymbol{l}_{\mathsf{ch}[n]}, \boldsymbol{l}_{(n,\mathsf{ch}[n])}), \quad n \in \boldsymbol{N}, \tag{1}$$

where $\boldsymbol{x}_{\mathrm{ch}[n]}$ and $\boldsymbol{l}_{\mathrm{ch}[n]}$ are the states and the labels of the nodes in $\mathrm{ch}[n]$, respectively, and $\boldsymbol{l}_{(n,\mathrm{ch}[n])}$ collects the labels of the edges connected to n. Different functions can be used for different node types k_n. For each node n, an output vector $\boldsymbol{o}_n$ is also defined that depends on the type k_n, the state $\boldsymbol{x}_n$ and the label $\boldsymbol{l}_n$ of the node. The dependence is described by a parametric *output function* $g_{\boldsymbol{w}}^{k_n}$

$$\boldsymbol{o}_n = g_{\boldsymbol{w}}^{k_n}(\boldsymbol{x}_n, \boldsymbol{l}_n), \quad n \in \boldsymbol{N}. \tag{2}$$

Thus, eqs. (1) and (2) define a method to compute an output $\boldsymbol{o}_n = \varphi_{\boldsymbol{w}}(\boldsymbol{G}, n)$ for each node n of the graph $\boldsymbol{G}$, taking into account the local descriptions and the relationships of all the nodes in $\boldsymbol{G}$. Each input graph $\boldsymbol{G}$ is evaluated on an *encoding network*, which is obtained by substituting all of the nodes of $\boldsymbol{G}$ with "units" that compute the function $f_{\boldsymbol{w}}^{k_n}$ and are connected according to graph topology. Each "f-unit" is also connected to another unit that implements the output function $g_{\boldsymbol{w}}^{k_n}$.

In this framework, a learning set $\mathcal{L}$ consists of triples $\mathcal{L} = \{(\boldsymbol{G}_i, n_{i,j}, \boldsymbol{t}_{n_{i,j}}) | \, 1 \leq i \leq p, \, 1 \leq j \leq q_i\}$, where each triple $(\boldsymbol{G}_i, n_{i,j}, \boldsymbol{t}_{n_{i,j}})$ denotes a graph $\boldsymbol{G}_i$, one of its nodes $n_{i,j}$ and the desired output $\boldsymbol{t}_{n_{i,j}}$. Moreover, p is the number of graphs in $\mathcal{L}$ and q_i is the number of the *supervised nodes* in graph $\boldsymbol{G}_i$, i.e. the nodes for which a desired output is defined. The goal of the learning procedure is to adapt the parameters $\boldsymbol{w}$ so that $\varphi_{\boldsymbol{w}}(\boldsymbol{G}, n_{i,j})$ approximates the target $t_{n_{i,j}}$ for the supervised nodes. In practice, learning is done by using gradient descent to minimize the quadratic error function

$$e_{\boldsymbol{w}} = \sum_{i=1}^{p} \sum_{j=1}^{q_i} (\boldsymbol{t}_{n_{i,j}} - \varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}))^2. \tag{3}$$

RelNNs and GNNs have a different implementation of $f_{\boldsymbol{w}}^{k_n}$ and $g_{\boldsymbol{w}}^{k_n}$ and a different computation of the states and the gradient. They also enforce additional constraints on the input data.

3.1 Relational neural networks

Relational neural networks [9, 10] are a special case of the above described framework with the following peculiarities:

1) The input graph $\boldsymbol{G}$ is acyclic and has a root node, from which there is a path to all other nodes. The root is the only supervised node of the graph.
2) The output function $g_{\boldsymbol{w}}^{k_n}$ is implemented by a layered feedforward neural network.
3) The transition function $f_{\boldsymbol{w}}^{k_n}$, which does not use the labels $\boldsymbol{l}_n$ and $\boldsymbol{l}_{(n,\mathrm{ch}[n])}$, is implemented by a recurrent neural network $r_{\boldsymbol{w}}^{k_n}$ that combines together the states and the labels of the children of a node, storing the result into an internal state $\boldsymbol{z}(i) \in \mathbb{R}^s$.

Point (1) is a fundamental property of RelNNs. Since the graph is acyclic, the corresponding encoding network is a large tree-like neural network whose components are the neural network units implementing $g_{\boldsymbol{w}}^{k_n}$ and $r_{\boldsymbol{w}}^{k_n}$. Thus, the states $\boldsymbol{x}_n$ and the gradient $\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$ can be computed with a common backpropagation algorithm. More precisely,

the states $\boldsymbol{x}_n$ are evaluated by the networks $r_{\boldsymbol{w}}^{k_n}$ from bottom to top: first the states of the nodes without children are calculated, then the states of their parents, and so on until the state of the root is obtained. Finally, the output is produced by $g_{\boldsymbol{w}}^{k_n}$. On the other hand, the gradient $\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$ is calculated by backpropagating the error from the root to the leaves (backpropagation through structure [5, 6]).

3.2 Graph neural networks

Graph neural networks have the following peculiarities:

1) Transition and output functions are the same for all nodes ($f_{\boldsymbol{w}} = f_{\boldsymbol{w}}^{k_n}$ and $g_{\boldsymbol{w}} = g_{\boldsymbol{w}}^{k_n}$).
2) The transition function $f_{\boldsymbol{w}}$ is implemented as a sum of contributions provided by all children. Each contribution is defined by the output of a feedforward neural network $h_{\boldsymbol{w}}$ that takes as input the labels of the node n, of one of its child and of the edge that links the two nodes

$$f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{x}_{\mathsf{ch}[n]}, \boldsymbol{l}_{\mathsf{ch}[n]}, \boldsymbol{l}_{(n,\mathsf{ch}[n])}) = \sum_{i=1}^{|\mathsf{ch}[n]|} h_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{x}_{\mathsf{ch}_i[n]}, \boldsymbol{l}_{\mathsf{ch}_i[n]}, \boldsymbol{l}_{(n,\mathsf{ch}_i[n])}). \quad (4)$$

Unlike RelNNs, GNNs can process cyclic graphs. Thus, three issues have to be addressed: (a) since the states $\boldsymbol{x}_n$ are defined recursively, depending on each other, it must be ensured that system (4) has a unique solution; (b) a method must be defined to compute the states $\boldsymbol{x}_n$; (c) an algorithm is required to compute the gradient $\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$.

In fact, let $\boldsymbol{F}_{\boldsymbol{w}}$ and $\boldsymbol{G}_{\boldsymbol{w}}$ be the vectorial functions obtained stacking all the instances of $f_{\boldsymbol{w}}$ and $g_{\boldsymbol{w}}$, respectively. Then eqs. (1) and (2) can be rewritten for GNN model as

$$\boldsymbol{x} = \boldsymbol{F}_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}), \quad \boldsymbol{o} = \boldsymbol{G}_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}), \quad (5)$$

where $\boldsymbol{l}$ represents the vector containing all the labels and $\boldsymbol{x}$ collects all the states. Issue (a) can be solved by designing $f_{\boldsymbol{w}}$ in such a way that the global function $\boldsymbol{F}_{\boldsymbol{w}}$ is a *contraction mapping* w.r.t. the state $\boldsymbol{x}$ (Banach fixed point theorem [11]). This goal can be achieved by adding a penalty term to the error function [12]. Banach fixed point theorem also suggests a method to solve issue (b). In fact, the theorem states that if ρ is a contraction mapping, then the dynamical system $\boldsymbol{x}(t+1) = \rho(\boldsymbol{x}(t))$, where $\boldsymbol{x}(t)$ denotes the t-th iterate of $\boldsymbol{x}$, converges exponentially fast to the solution of the equation $\boldsymbol{x} = \rho(\boldsymbol{x})$ for any initial state $\boldsymbol{x}(0)$. Eq. (5) can therefore be solved by iteration.

Finally, in order to design a learning algorithm, we can observe that the encoding network represents a system having a settling behaviour. For this reason, the gradient can be computed using Almeida–Pineda algorithm [13, 14], in combination with a backpropagation through structure algorithm.

4 Experimental results

In this section, GNNs and RelNNs are evaluated experimentally on some relational datasets. The considered problems include the modeling of queries produced by aggregate functions and two datasets dealing with QSAR problems that are often used to compare machine learning techniques.

Table 1. Accuracies on the aggregate function experiments. Results are averages over 3 runs.

Problem	GNN	RelNN
count	100	100
sum	100	100
max	62.8	45.9
avg	92.4	99.7
median	79.0	85.5

Aggregate functions Modeling an aggregate function can be considered the minimal requirement for a relational learner. An aggregate function is applied to a bag of tuples and produces one tuple with one attribute that contains the result of the operation. Thus, for each function, data may be represented as a simple graph with one node (the result tuple) that is connected to all the other nodes (the tuples of the bag). Our datasets contained 500 random bags: 300 in the training set, 100 in validation set and 100 in test set. Each bag included 5 to 10 tuples, while each tuple had 5 random real attributes in the interval $[-0.8, 0.8]$. The aggregate function to be learned considers only one attribute, the others are noise. Table 1 shows the results obtained for the aggregate functions count, sum, maximum, average and median. Each bag is considered correctly predicted if the output of the network is within ± 0.1 range of the target.

The results show that the performances achieved by the two connectionist models vary largely according to the considered aggregate function. This variance is partially due to the general difficulty of neural networks to approximate some kind of functions. Also, RelNNs combine the contributions of the children using a recurrent neural network that process a child at every time instance. If the input sequence is long, the recurrent neural network tends to forget the initial inputs. For this reason, the performance of RelNNs is worse on the maximum function, where the ability to remember a value is more critical. This is only one tentative explanation, further work is needed to understand the different behavior of both approaches.

The mutagenesis dataset The mutagenesis dataset [17] is a small dataset, publicly available and often used as a benchmark in ILP literature [15]. It contains the descriptions of 230 molecules and the goal is to predict which compounds are mutagenic. In [17] it is shown that 188 molecules out of 230 are learnable using linear regression. This subset was therefore called "regression–friendly", while the remaining 42 compounds were termed "regression–unfriendly". Many authors have reported their results only on the "regression–friendly" part, often referred to as "the" mutagenesis dataset.

Each compound is provided with four global features (C+PS). Moreover, the atom-bond structure (AB) is also given which defines binary relationships between the atoms of each compound. Two graphical representations are possible:

1) Each compound is represented by a node labeled with the global features and connected to other nodes denoting the atoms that belong to the compound. Atom nodes are labeled with the type of the atom and are connected by edges to other nodes, according to the atom–bond structure.

Table 2. A comparison of the performance of GNNs, RelNNs and other techniques (results from [15] and [16]) on the mutagenesis (regression-friendly, regression-unfriendly and whole dataset). FG denotes functional group features that have been be used as higher level features.

Method	Knowledge	Friendly	Unfriendly	Whole
GNN	AB+C+PS	94.3 ± 0.6	96.0 ± 1.4	90.5 ± 0.7
RelNN	AB+C+PS	91.0 ± 1.8	86.8 ± 2.6	86.5 ± 2.1
Neural Networks	C+PS	89.0 ± 2.0		
P-Progol	AB+C	82.0 ± 3.0		
P-Progol	AB+C+FG	88.0 ± 2.0		
MFLOG	AB+C	95.7		
FOIL	AB	76		
boosted-FOIL	not available	88.3		
$1nn(d_m)$	AB	83 ± 2.0	72 ± 7.0	
$1nn(d_m)$	AB+C	91 ± 2.0	72 ± 7.0	
RDBC	AB	84	79	83
RDBC	AB+C	83	79	82
RSD	AB+C+FG	92.6		
SINUS	AB+C+FG	84.5		
RELAGGS	AB+C+FG	88.0		
RS	AB	88.9 ± 7.2		
RS	AB+FG	89.9 ± 5.2		
RS	AB+C+PS+FG	95.8 ± 3.3		
TILDE	AB		85	77
TILDE	AB+C		79	82

2) The compound is not represented. Nodes representing atoms are connected as in (1), while their labels are extended with the global features.

Our experimentation has been carried out using (1) for RelNNs and (2) for GNNs. To evaluate the results, we adopted a 10-fold cross-validation scheme and averaged the accuracies over 5 runs. Results are shown in table 2.

GNNs clearly outperform other methods on the whole dataset and on the unfriendly part. RelNNs produce a slightly lower result. It is also interesting to note that whereas most of the approaches show a higher level of accuracy on the whole dataset than on the unfriendly part, the converse holds for our approaches. Such a behaviour may suggest that the proposed connectionist models can capture particular features of the dataset, which cannot be captured by other methods and are distributed more homogeneously inside each single part (friendly and unfriendly) than between the parts.

The biodegradability dataset The biodegradability dataset, introduced by Dzeroski et al. [18], is very similar to the mutagenesis dataset. The aim is to classify 328 chemical compounds according to their degree of biodegradability: resistant, slow, moderate or fast. Five different representations can be used for learning. Three of them are global: P0, which includes molecular weight and logP, P1, which consists of counts of the different types of functional groups in the molecule, and P2, which consists of counts of common substructures in the molecules. The other two regard the atoms: R0, containing atoms and bonds descriptions, and R1, including background predicates about

Table 3. Results for biodegradability. First six rows are our results, the other results are reported in [18]. The accuracy (+/-1) column gives the number of examples for which the classification is maximum one class up or down from the true classification.

Method	Knowledge	Accuracy	Accuracy (+/-1)
GNN	P0+P1+P2+R0	54.7	91.7
GNN	P0+P1+R0	53.5	89.9
GNN	P0+R0	50.6	91.0
RelNN	P0+P1+P2+R0	57.6	92.1
RelNN	P0+R0	48.6	91.5
Feedforward NN	P0	43.7	89.0
C4.5	P0+P1	55.2	86.2
C4.5	P0+P2	56.9	82.4
RIPPER	P0+P1	52.6	89.8
RIPPER	P0+P2	57.6	93.9
M5'	P0+P1	53.8	94.5
M5'	P0+P2	59.8	94.7
FFOIL	P0+R0	53.0	88.7
ICL	P0+R1	55.7	92.6
SRT-C	P0+P1	50.8	87.5
SRT-C	P0+P1+R1	55.0	90.0
SRT-R	P0+P1	49.5	91.9
SRT-R	P0+P1+R1	51.6	92.8
TILDE-C	P0+R1	51.0	88.6
TILDE-C	P0+P1+R1	52.0	89.0
TILDE-R	P0+R1	52.6	94.0
TILDE-R	P0+P1+R1	52.4	93.9

functional groups and substructures. Representation and settings for our methods were similar to the mutagenesis experiment. Table 3 shows results for different methods.

The results for our models are comparable to the other results, but not better. This means that counts of functional groups and substructures are probably good propositionalizations of the relational data, as already suggested in [18]. When P1 and P2 are left out, the performance of GNNs and RelNNs decreases. However, in this case the performance is still better than when using the global properties P0 alone, so at least part of this information seems to be learned from the simple connectivity alone. One of the reasons why it is more difficult to learn the substructures from the simple atoms and bonds descriptions, could be that some of them are rather infrequent in the data.

5 Conclusions

In this paper, two recently connectionist models, i.e. Relational Neural Networks (RelNNs) and Graph Neural Networks (GNNs), were experimentally evaluated and compared. The results on the mutagenesis and the biodegradability datasets were promising and suggested that RelNNs and GNNs can be viable approaches for learning on relational data. In particularly, GNNs achieved the current best result on mutagenesis. The experimentation of the models on other and more complex benchmarks and a comparative study of their theoretical properties is matter of future research.

References

1. Bianchini, M., Maggini, M., Sarti, L., Scarselli, F.: Recursive neural networks for processing graphs with labelled edges: Theory and applications. Neural Networks - Special Issue on Neural Networks and Kernel Methods for Structured Domains **18** (2005) 1040–1050
2. Francesconi, E., Frasconi, P., Gori, M., Marinai, S., Sheng, J., Soda, G., Sperduti, A.: Logo recognition by recursive neural networks. In Tombre, K., Chhabra, A.K., eds.: GREC '97: Selected Papers from the Second International Workshop on Graphics Recognition, Algorithms and Systems. Springer-Verlag (1998) 104–117
3. Kondor, R., Lafferty, J.: Diffusion kernels on graphs and other discrete structures. In Sammut, C., Hoffmann, A.e., eds.: Proc. 19th International Conference on Machine Learning (ICML2002), Morgan Kaufmann Publishers Inc (2002) 315–322
4. Gärtner, T.: Kernel-based learning in multi-relational data mining. ACM SIGKDD Explorations **5**(1) (2003) 49–58
5. Sperduti, A., Starita, A.: Supervised neural networks for the classification of structures. IEEE Transactions on Neural Networks **8** (1997) 429–459
6. Frasconi, P., Gori, M., Sperduti, A.: A general framework for adaptive processing of data structures. IEEE Transactions on Neural Networks **9**(5) (1998) 768–786
7. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. In: Proc. International Joint Conference on Neural Networks (IJCNN2005). (2005) 729–734
8. Hagenbuchner, M., Sperduti, A., Tsoi, A.C.: A self-organizing map for adaptive processing of structured data. IEEE Transactions on Neural Networks **14**(3) (2003) 491–505
9. Blockeel, H., Bruynooghe, M.: Aggregation versus selection bias, and relational neural networks. IJCAI-2003 Workshop on Learning Statistical Models from Relational Data, SRL-2003, Acapulco, Mexico, August **11** (2003)
10. Uwents, W., Blockeel, H.: Classifying relational data with neural networks. In: Proc. of the 15th International Conference on Inductive Logic Programming, Bonn, Germany (2005)
11. Khamsi, M.A.: An Introduction to Metric Spaces and Fixed Point Theory. John Wiley & Sons Inc (2001)
12. Scarselli, F., Gori, M., Monfardini, G., Tsoi, A.C., Hagenbuchner, M.: A new neural network model for graph processing. Technical Report DII 01/05, Department of Information Engineering, University of Siena (2005)
13. Almeida, L.: A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In Caudill, M., Butler, C., eds.: Proceedings of the IEEE International Conference on Neural Networks. Volume 2., San Diego, 1987, IEEE, New York (1987) 609–618
14. Pineda, F.: Generalization of back–propagation to recurrent neural networks. Physical Review Letters **59** (1987) 2229–2232
15. Lodhi, H., Muggleton, S.H.: Is mutagenesis still challenging? In: Proceedings of the 15th International Conference on Inductive Logic Programming, ILP 2005, Late-Breaking Papers. (2005) 35–40
16. De Raedt, L., Blockeel, H.: Using logical decision trees for clustering. In: Proceedings of the 7th International Workshop on Inductive Logic Programming ILP 1997. Volume 1297 of Lecture Notes in Artificial Intelligence., Springer-Verlag (1997) 133–141
17. Debnath, A., Lopex de Compandre, R., Debnath, G., Schusterman, A., Hansch, C.: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. Journal of Medicinal Chemistry **34**(2) (1991) 786–797
18. Dzeroski, S., Blockeel, H., Kompare, B., Kramer, S., Pfahringer, B., Van Laer, W.: Experiments in predicting biodegradability. Proceedings of the Ninth International Workshop on Inductive Logic Programming **1634** (1999) 80–91

Edgar: the Embedding-baseD GrAph MineR

Marc Wörlein,[1] Alexander Dreweke,[1] Thorsten Meinl,[2] Ingrid Fischer[2], and Michael Philippsen[1]

[1] University of Erlangen-Nuremberg, Computer Science Department 2
Martensstr. 3, 91058 Erlangen, Germany
{woerlein,dreweke,philippsen}@cs.fau.de
[2] ALTANA Chair for Bioinformatics and Information Mining
University of Konstanz, BOX M712, 78457 Konstanz, Germany
{Thorsten.Meinl,Ingrid.Fischer}@inf.uni-konstanz.de

Abstract. In this paper we present the novel graph mining algorithm Edgar which is based on the well-known gSpan algorithm. The need for another subgraph miner results from procedural abstraction (an important technique to reduce code size in embedded systems). Assembler code is represented as a data flow graph and subgraph mining on this graph returns frequent code fragments that can be extracted into procedures. When mining for procedural abstraction, it is not the number of data flow graphs in which a fragment occurs that is important but the number of all the non-overlapping occurrences in all graphs. Several changes in the mining process have therefore become necessary. As traditional pruning strategies are inappropriate, Edgar uses a new embedding-based frequency; on average, saves 160% more instructions compared to classical approaches.

1 Introduction

Nowadays embedded systems are used in many fields such as automotive, cellphones, and various consumer electronics. As consumers demand more and more functionality, the programs that run on these systems grow. To reduce the per piece costs the manufacturers have to reduce the memory, space, and energy requirements. So they invest a great deal of effort in reducing the amount of code. Procedural abstraction (PA) is the most important technique to deal with code repetitions: frequent code fragments are extracted and substituted with jumps or procedure calls. Only one fragment remains in the resulting code.

This paper demonstrates that frequent graph mining can be applied successfully to enhance procedural abstraction. This section briefly introduces procedural abstraction and frequent graph mining. Section 2 discusses related work. Section 3 describes our new subgraph miner satisfying all requirements for procedural abstraction in detail. An evaluation is given in section 4.

1.1 Procedural Abstraction

Although there are compiler flags to avoid code bloat and smart linkers that reduce code size, still many space-wasting code duplications remain in the com-

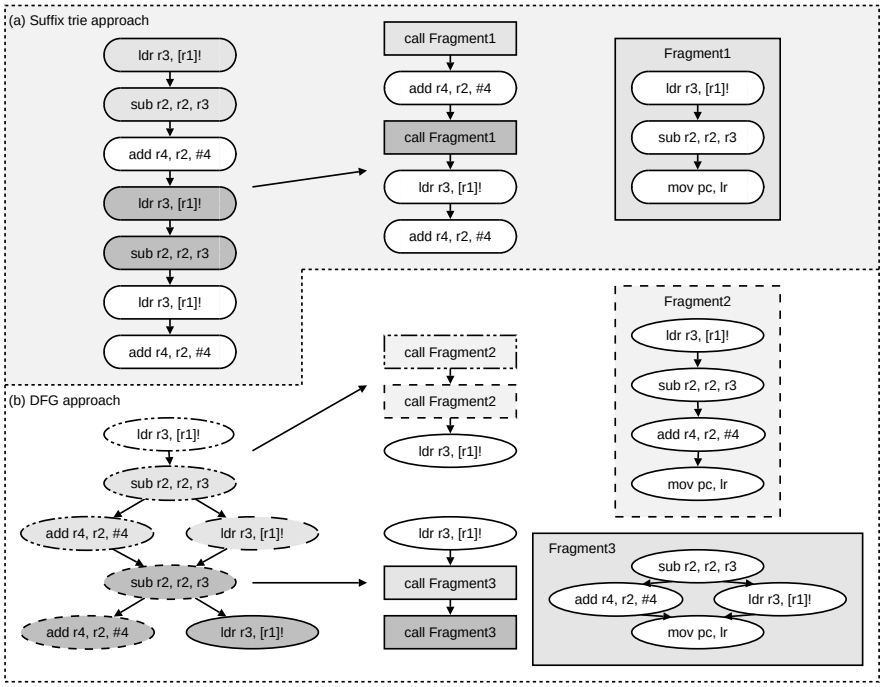


Fig. 1: Running PA example: (a) sequencial PA on a basic block of ARM code and (b) the corresponding data flow graph extractions.

piled executables [4] (resulting from code templates, copy-and-paste programming, etc.). Code-size optimization is important for embedded systems [1] as cost and energy consumption depends on the size of the built-in memory and because more functionality fits into memory of a given size.

The main focus of code compaction based on PA is to detect repeated code fragments that can be extracted afterwards. Early compaction approaches considered the whole code as a sequence of instructions and used suffix tries to detect common subsequences [7]. Therefore fast algorithms that keep the compile-time short are used. For embedded systems, a larger time budget is available since the cost per piece is more relevant than compile time.

Optimizing compiliers have passes that reorder the code sequence to e.g. keep the instruction pipeline perfectly filled. Therfore the traditional linear suffix trie approach cannot detect all semantically equivalent code, because the instruction order may differs from occurrence to occurrence. Our new graph-based PA approach transforms the instruction sequences of basic blocks[3] into data flow graphs (DFG) which are independent of the actual scheduling.

[3] A basic block is a piece of code that ensures that if one instruction is executed all following instructions will also be executed under all circumstances.

We explain the basic ideas of our approach to PA with the (synthetic) piece of ARM code and its corresponding DFG shown on the left side of Fig. 1. Our example steps through the values of an array (register `r1` points to the current element in the array) and performs some calculations. The upper part of Fig. 1 shows how the suffix trie based approaches detect the code fragment `ldr r3,[r1]!` → `sub r2,r2,r3` twice. The lower part shows that graph-based PA is more successful, even for this tiny example, because it finds two different fragments of size three that both appear twice and that both result in a smaller number of instructions. The varying order of instructions prevents suffix tries from detecting these fragments.

1.2 Frequent Subgraph Mining

Several frequent subgraph mining algorithms have been published that allow graph databases to be searched for frequent graph fragments by traversing the lattice of subgraphs [14]. An example of such a search lattice is given in Fig. 2. The empty subgraph (*) at the top is followed by the subgraphs with just one node (i.e. 1 instruction). Each step downwards adds a new edge (and node, if necessary) to a subgraph. Real search lattices are much more complex than this example as the underlying databases consist of many different and much bigger graphs.

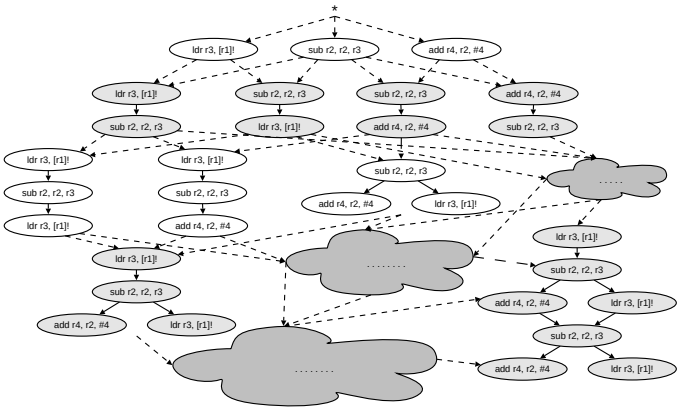


Fig. 2: Parts of the search lattice for the example.

A graph fragment is usually considered *frequent* if it appears in a given number of database graphs. Only the number of database graphs in which it occurs is counted, no matter how often a fragment is found in a single database graph.

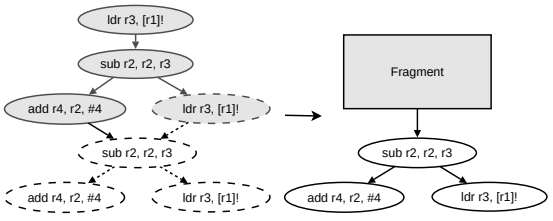


Fig. 3: Two overlapping embeddings

This counting scheme is inappropriate for PA where the total number of occurrences (called *embeddings*) of the graph fragment matters. In Fig. 1 the frequent fragment has a frequency of 1 since it appears twice but only in one graph.

However, not all embeddings are relevant for PA. If 2 embeddings overlap (as in Fig. 3) only the code for one of them can be extracted by PA. When one is extracted, the other embedding disappears. So only the number of non-overlapping embeddings matters.

2 Related Approaches

Within the graph mining community only a few algorithms have been published that satisfy the requirements of PA. The well-known algorithms such as MoFa, gSpan, Gaston, or FFSM [15] use inappropriate graph-based frequency counting. Among the embedding-based algorithms, there are heuristic algorithms such as SubDue [3], GBI [12], or SEuS [8]. In contrast to Edgar, they only check heuristic subsets of fragments, and do not find the optimal candidate in general. The algorithms of Vanetik [13] and Kuramochi [11] are similar to our approach as they calculate the maximal independent set to determine edge-disjoint embeddings. But since two edge-disjoint embeddings can share a node (i.e. one instruction in our application) these algorithms are unsuitable for PA.

In the area of circuit synthesis on FPGAs (field-programmable gate arrays) similar problems arise [17]. High-level synthesis compilers produce recurring patterns that can be extracted to reduce the number of functional units. This so called *template generation* is similar to frequent graph mining. But there seems to be no attempt to do a complete detection as in our approach. Similar to us, [2, 12, 17] use a data flow graph-based approach to identify frequent patterns for a hard-logic implementation on embedded systems. However, their heuristics or restrictions to fragments with just one source node[4] do not ensure that all relevant patterns are detected.

3 Edgar (Embedding-baseD GrAph mineR)

For mining DFGs we have developed a new mining algorithm called Edgar that is based on the well known gSpan algorithm [16]. We decided to start from gSpan because it is the best algorithm for mining with graph-based frequency [15].

3.1 gSpan

Fig. 2 shows parts of a search lattice. There are multiple paths to most subgraphs, which have to be filtered out to avoid duplicate detection of the corresponding fragments. Therefore gSpan builds a so-called *dfs-code* that represents the edges in the order in which they are added to the fragment. As there are different paths to each fragment, different codes exist. A global order on these codes allows the smallest one to be defined as canonical [16]. gSpan only extends fragments with canonical representation, because the non-canonical ones have already been created before.

[4] A source node has just outgoing and no incoming edges.

To detect non-canonical codes, a straightforward but expensive graph isomorphism test can be used. To minimize these tests, gSpan groups all possible extending edges into forward edges that insert a new node, and backward edges that just close cycles. Now, gSpan only extends fragments with forward edges starting at nodes on the *right most path*[5] and backward edges starting at the last added node. This heuristic method eliminates many unnecessary paths. Canonical tests are only required for the paths that remain.

For each fragment gSpan only stores a list of database graphs in which the fragment appears (a so-called *appearance list*). Its size represents the frequency of the fragment. To extend a fragment, only the graphs of that list have to be scanned (as opposed to the whole dataset).

3.2 Extending gSpan for PA

For Edgar, we have adopted the basic structures of the search lattice and gSpan's basic dfs-code including the right most extension rule. However, since the original gSpan algorithm works on undirected graphs, we had to extend the dfs-code, the extension process, and the test for being canonical to work on directed graphs needed for PA.

Edgar's frequency-based pruning significantly differs from gSpan's. For graph-based miners, the frequency of a fragment decreases monotonically with growing fragment size. Hence the search can be pruned as soon as an infrequent fragment is found. For embedding-based pruning *all* embeddings and not only the supported graphs are relevant. Therefore, first all embeddings have to be stored (not just the appearance lists). This leads to higher memory requirements.[6] Second, since there are cases in which the frequency *increases* with growing fragment size (see Fig. 4) a corresponding frequency pruning is wrong.

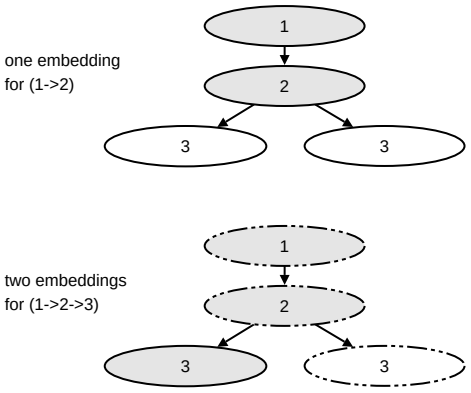


Fig. 4: Embedding counts.

Fortunately for PA it is not the set of all embeddings that matters, but the non-overlapping subset (see section 1.2). The ones with maximal size (the maximal number of embeddings) are the best, because extracting the embeddings of such maximal non-overlapping sets results in best code compression. Additionally, the maximal size decreases monotonically as the graph-based frequency does.

[5] This is the path of forward edges, between the first and the last inserted node.
[6] It is better to keep embeddings rather than recomputing occurrences.

This is obvious, because a non-overlapping subset for each fragment can be generated by the embeddings of one of its successors. For each parent a non-overlapping subset with the same size can be generated out of the maximal subset of a child.[7]. This subset does not need to be a maximal subset, which therefore can contain the same number or more embeddings. For each successor such a subset can be found, thus the *maximal* subset for each fragment is greater than or equal to the sets of its children, which allows pruning like the original frequency.

Edgar builds a so-called collision graph to determine the size of a maximal non-overlapping subset. In the collision graph the embeddings of a fragment are represented as nodes which are connected if the corresponding embeddings overlap. A maximum independent set in the collision graph then represents a subset of maximal non-overlapping embeddings on the graph database. Alternatively, a maximum clique in the inverted collision graph[8] also represents the desired subset. For computing the maximum independent set/maximum clique – which is an NP-complete problem – standard algorithms (like [6]) can be used. Edgar adapts the maximal clique algorithm of Kumlander [10] which at the moment is the fastest known algorithm.

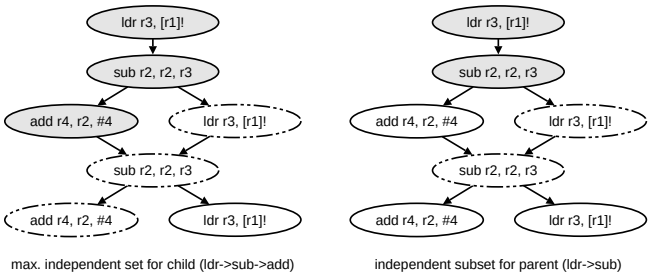


Fig. 5: A parent's subset induced by a child's maximal one.

4 Evaluation

We have evaluated our approach with a subset of the MiBench suite [9]. For embedded systems it is sufficient to statically link against the small *dietlibc*[9] [5]. Additionally all binaries were compiled with `-Os` to optimize for size. Table 6 compares the traditional suffix trie approach (SFX) with DgSpan, the directed graph-based gSpan algorithm, and with the embedding-based Edgar. Repeatedly for each

Program	# Inst.	# of saved inst.		
		SFX	DgSpan	Edgar
bitcnts	3946	53	81	83
crc	3584	46	68	119
dijkstra	4632	70	102	164
patricia	5039	70	105	189
qsort	4770	70	105	197
rijndael	7113	70	132	256
search	3717	46	74	110
sha	3897	55	82	120
total	36698	480	749	1238

Fig. 6: Saved instructions

[7] In the example in Fig. 5 only the unused `add r4, r2, #4` has to be removed
[8] The inverted collision graph has an edge, if two nodes do not overlap
[9] Dietlibc is a cross platform C run-time library, supporting x86, ARM, Sparc, Alpha, PPC, Mips, and s390 architectures, compatible with SUVv2 and POSIX.

approach, in each iteration the best fragment has been extracted until no further shrinking was possible. The fragments are weighted by the number of instructions they save. In total, Edgar saves 1238 as opposed to 480 instructions by SFX which is an improvement by a factor of 2.6. Edgar even saves more than DgSpan, because DgSpan misses many fragments that appear several times in the same basic block.

From the related work (section 2) only SubDue is available for download. Therefore we have simply compared Edgar and SubDue quantitatively. In both algorithms we used the same weighting function $(graphSize - 1) * (frequency - 1)$ to estimate the instruction savings for a given fragment. For Edgar the support is set to two to find the overall best fragments. A real substitution of the found fragment is not trivial, so just the first run for both algorithms is taken for the results in Fig. 7. For the comparison ten different random graphs with uniformly distributed node and edge labels[10] - five with the parameters of the first column of the given table and the other half with the second values - are generated with the *subgen* tool of the SubDue-system. Fig. 7 shows the average weight over the best three fragments (default configuration of SubDue) for each algorithm. In most cases SubDue found the same fragments as Edgar but not the optimal number of independent embeddings, which lead to lesser ranking of those fragments.

edges	200	300
Connectivity	3	2
Coverage	0.7	0.8
Overlap	0.7	0.2
Deviation	0.9	1.0
SubVertexLabel	0.30	0.25
SubEdgeLabel	0.20	0.25
DelVertex	0.15	0.10
DelEdge	0.35	0.40
Substruct. nodes	3	6
Substruct. edges	6	5

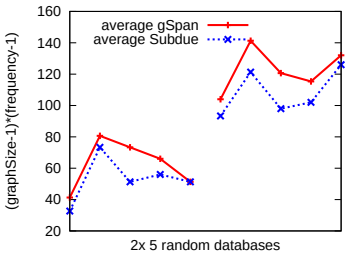


Fig. 7: SubDue vs. gSpan: best found fragments

5 Conclusion

Frequent graph mining renders PA more effective and helps to build cheaper and/or more powerful embedded systems. By using DFGs instead of code sequences as a basis of frequent code detection more potential for outlining is found. The better the candidates are with respect to size and frequency, the smaller the resulting code. Compared to other sequential approaches our prototype Edgar saves an average of 160 % more instructions.

In contrast to Edgar, common graph-based miners cannot be applied, since they do not work on directed graphs, they use heuristics that prune the search space prematurely, stumble over fragments that have nodes in common, or lack embedding-based frequency counting. The paper presented a new embedding-based frequency definition that is based on the maximal non-overlapping subset of all embeddings and results in a valid pruning strategy.

[10] 25 node and 4 edge labels

References

1. J. Bentley. Programming Pearls: Squeezing Space. *Communications of the ACM*, 27(5):416–421, May 1984.
2. P. Brisk, A. Kaplan, R. Kastner, and M. Sarrafzadeh. Instruction generation and regularity extraction for reconfigurable processors. In *Proc. of the Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 262–269, New York, 2002.
3. D. J. Cook and L. B. Holder. Substructure Discovery Using Minimum Description Length and Background Knowledge. *Artificial Intelligence Research*, 1:231–255, 1994.
4. S. K. Debray, W. Evans, R. Muth, and Bjorn de Sutter. Compiler Techniques for Code Compaction. *ACM Trans. Progamming Languages and Systems*, 22(2):378–415, 2000.
5. dietlibc - a libc optimized for small size. http://www.fefe.de/dietlibc/.
6. F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and Conquer: A Simple $o(n^{0.288n})$ Independent Set Algorithm. In *Proc. of 17th ACM-SIAM Symp. on Discrete Algorithms*, pages 18–25, Miami, FL, January 2006.
7. C. W. Fraser, E. W. Myers, and A. L. Wendt. Analyzing and Compressing Assembly Code. In *Proc. ACM Symp. an Compiler Construction*, pages 117–121, Montréal, C, 1984.
8. S. Ghazizadeh and S. S. Chawathe. SEuS: Structure extraction using summaries. In *Discovery Science*, volume 2534, pages 71–85. Springer, 2002.
9. M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proc. 4th IEEE Workshop on Workload Characterization*, pages 3–14, Austin, TX, 2001.
10. D. Kumlander. A new exact Algorithm for the Maximum-Weight Clique Problem based on a Heuristic Vertex-Coloring and a Backtrack Search. In *Proc. 5th Int'l Conf. on Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 202–208, Metz, France, July 2004.
11. M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data Min. Knowl. Discov.*, 11(3):243–271, November 2005.
12. P. C. Nguyen, K. Ohara, H. Motoda, and T. Washio. Cl-gbi: A novel approach for extracting typical patterns from graph-structured data. In *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference Proc.*, volume 3518, pages 639–649, Hanoi, Vietnam, May 2005. Springer.
13. N. Vanetik, E. Gudes, and S. E. Shimony. Computing frequent graph patterns from semistructured data. In *Proc. IEEE Int'l Conf. on Data Mining ICDM*, page 458, Maebashi City, Japan, November 2002.
14. T. Washio and H. Motoda. State of the Art of Graph–based Data Mining. *SIGKDD Explorations Newsletter*, 5(1):59–68, July 2003.
15. M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen. A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and Gaston. In *Knowledge Discovery in Database: PKDD 2005*, volume 3721, pages 392–403, Berlin, 2005. Springer.
16. X. Yan and J. Han. gSpan: Graph–Based Substructure Pattern Mining. In *Proc. IEEE Int'l Conf. on Data Mining ICDM*, pages 721–723, Maebashi City, Japan, November 2002.
17. D. Zaretsky, G. Mittal, R. P. Dick, and P. Banerjee. Dynamic template generation for resource sharing in control and data flow graphs. In *19th Int' Conf. on VLSI Design*, pages 465–468, Hyderabad, India, January 2006.

Author Index