Erschienen in: Algorithms and Discrete Applied Mathematics : Third International Conference, CALDAM 2017 ; Proceedings / Gaur, Daya; Narayanaswamy, N.S. (Hrsg.). - Cham : Springer, 2017. - (Lecture Notes in Computer Science ; 10156). - S. 60-71. - ISSN 0302-9743. - eISSN 1611-3349. - ISBN 978-3-319-53006-2
https://dx.doi.org/10.1007/978-3-319-53007-9_6

Positional Dominance: Concepts and Algorithms

Ulrik Brandes, Moritz Heine, Julian Müller(✉), and Mark Ortmann

Computer and Information Science, University of Konstanz, Konstanz, Germany
Julian.Mueller@uni-konstanz.de

Abstract. Centrality indices assign values to the vertices of a graph such that vertices with higher values are considered more central. Triggered by a recent result on the preservation of the vicinal preorder in rankings obtained from common centrality indices, we review and extend notions of domination among vertices. These may serve as building blocks for new concepts of centrality that extend more directly, and more coherently, to more general types of data such as multilayer networks. We also give efficient algorithms to construct the associated partial rankings.

1 Introduction

One of the core concepts of network analysis is the identification of central vertices [13]. The most commonly applied centrality indices measure, e.g., the number of vertices a vertex can communicate with directly (*degree*), the expenses of a vertex to reach each other vertex in the network (*closeness* [21]), and the control over communication of others in the network (*betweenness* [6]).

While all centrality indices assign numerical values to each vertex in the graph, one is typically only interested in the derived ranking. Although well established centrality indices differ substantially in their definition, the rankings they induce all coincide on the vicinal preorder. In the *vicinal preorder* [5], a vertex $w \in V$ dominates another vertex $v \in V$, i.e. $v \leq w$, if and only if $N(v) \subseteq N[w]$ where $N(u)$ is the neighborhood of vertex u in the graph and $N[u] = N(u) \cup \{u\}$. This implies that it is possible to construct a partial ranking of the vertices by simply comparing their neighborhoods, and this ranking is preserved by any centrality index [22].

The vicinal preorder, or *neighborhood inclusion*, is itself the union of two other preorders: (i) the *dominance preorder* where $v \leq_a w \iff N[v] \subseteq N[w]$ and (ii) the *structural preorder* where $v \leq_n w \iff N(v) \subseteq N(w)$. Furthermore, it is an instantiation of *positional dominance* [1], a generic concept that allows for valued relationships and the expression of levels of homogeneity, i.e., admissible substitutions of vertices in the comparison of neighborhoods. Positional dominance provides a building block on which concepts of centrality can not only be generalized more easily, but also more coherently, to more complex kinds of data. While we are motivated by the implications of variant preorders for centrality, we are especially interested in their computational complexity here.

Contribution. We present efficient algorithms for instances of positional dominance. Our main contribution is an algorithm with $\mathcal{O}(nm \log\log \Delta(G))$ running time for the homogeneous case with weights on both edges and vertices. This is an improvement over the straightforward approach with an $\mathcal{O}(nm\Delta(G)^{3/2})$ time bound. In addition we give lower bounds for worst-case running times by constructing families of graphs with large output size, i.e., dense preorders. Although we consider simple undirected graphs, our results can be adapted for weighted, directed, and graphs with a given bipartition (*two-mode graphs*).

Note, however, that we assume throughout this paper that our input graphs do not contain isolated vertices because these are dominated by every other vertex in the graph (or no other vertex in the dominance preorder), so that their relationships can be checked in constant time and are best represented implicitly.

2 Preliminaries

For the most part, we consider simple undirected graphs $G = (V, E)$, where both vertices and edges may carry weights $\omega : V \cup E \to \mathbb{R}$. For edges $\{v, w\} \in E$, we use shorthand notation $\omega(v, w) = \omega(\{v, w\})$. Weights can be thought of as non-negative reals for convenience but any ordered range of values will do. Following the usual convention, we denote the number of vertices and edges by $n = n(G) = |V|$ and $m = m(G) = |E|$. We write $H \subseteq G$ if H is a subgraph of G, and $G[W]$ for the subgraph induced by $W \subseteq V$.

The (open) *neighborhood* of a vertex $v \in V$ is defined as $N(v) = \{w : \{v, w\} \in E\}$ and the *closed neighborhood* as $N[v] = N(v) \cup \{v\}$. We assume that $N(v) \neq \emptyset$, $v \in V$, throughout the paper. The *degree* of $v \in V$ is $\deg(v) = |N(v)|$, and since $2m = \sum_{v \in V} \deg(v)$ the *average degree* is $\langle \deg \rangle = \frac{2m}{n}$. Let $\Delta(G) = \max\{\deg(v) : v \in V\}$ denote the *maximum degree* of a graph.

The *arboricity* $\alpha(G)$ of a graph G is the minimum number of forests needed to cover its edges. Arboricity is an indicator of sparseness as it is closely related to the average degree in a densest subgraph via $\alpha(G) = \max_{H \subseteq G}\{\lceil \frac{m(H)}{n(H)-1} \rceil\}$ [16].

A binary relation $R \subseteq (V \times V)$ is called a *preorder* if it is reflexive and transitive. Since a total preorder gives a ranking, we may refer to a preorder also as a *partial ranking*. If a (partial) ranking is antisymmetric, it is a (partial) order.

3 Dominance

The dominance preorder is a restriction of the more general vicinal preorder. A vertex $w \in V$ *(vertex) dominates* a vertex $v \in V$, $v \leq_a w$, if $N[v] \subseteq N[w]$. The subscript indicates that a relation w.r.t. the dominance preorder can only exist for pairs of adjacent vertices. Any two vertices that dominate each other are also referred to as *true twins*, because they are adjacent and have exactly the same neighborhood. Consequently, each equivalence class of the dominance relation $\leq_a$ induces a clique.

Algorithm 1: Dominance Preorder

input : simple undirected graph $G = (V, E)$
output : partial ranking $\leq_a$ on V (dominance)

initialize $\leq_a$ with $v \leq_a v$ for all $v \in V$;
for $\{v, w\} \in E$ **do** $\deg(v, w) \leftarrow 0$
for $v_i = v_1, \ldots, v_n$ *where* $\deg(v_1) \geq \ldots \geq \deg(v_n)$ **do**
  mark all $w \in N^+(v_i)$ with v_i;
  for $w \in N^+(v_i)$ **do**
    for $u \in N^+(w)$ **do**
      if u *is marked with* v_i **then**
        foreach $\{i, j\} \in \binom{\{v_i, w, u\}}{2}$ **do** increment $\deg(i, j)$
    if $\deg(v_i, w) = \deg(v_i) - 1$ **then** add $v_i \leq_a w$
    if $\deg(v_i, w) = \deg(w) - 1$ **then** add $w \leq_a v_i$

To construct the dominance preorder, we extend the concept of neighborhood to edges $\{v, w\} \in E$ via $N(v, w) = N(v) \cap N(w)$, and denote $\deg(v, w) = |N(v, w)| < \min\{\deg(v), \deg(w)\}$. Since $\deg(v, w) = \deg(v) - 1$ means every neighbor of v other than w itself is also a neighbor of w, it implies $v \leq_a w$. Thus, the dominance preorder can be determined using any algorithm that counts the number of triangles an edge is part of.

Algorithm 1 is based on an efficient realization [17] of the triangle listing algorithm of Chiba and Nishizeki [2]. Given any ordering of the vertices, here specifically from higher to lower degrees, we let $N^+(v)$ denote the number of adjacent vertices that appear after v in the ordering, i.e., all edges are oriented from earlier to later respective to the ordering.

Theorem 1. *Algorithm 1 determines the dominance preorder of a simple undirected graph in time* $\mathcal{O}(\alpha(G)m)$.

Proof. The vertex ordering ensures that for each edge, the neighbors of the vertex with smaller degree are inspected. Following Chiba and Nishizeki's reasoning for their algorithm `K3` [2], the claimed runtime is a consequence of the inequality

$$\sum_{\{u,v\} \in E} \min\{\deg(u), \deg(v)\} \;\leq\; 2\alpha(G)m$$

□

As the arboricity of a graph can be as large as n, Algorithm 1's running time is far from being linear in the size of the input and output, since clearly the size of the dominance preorder is bounded from above by m.

However, although in the worst case the arboricity is linear in n, it is often small in social networks [4] and can, in fact, be even smaller than the average degree [2]. We show next that there is no simple relationship between these two graph invariants because arboricity is determined by the densest subgraph.

Theorem 2. *There is a family of graphs G_n, $n > 3$, with $\langle \deg \rangle \in \mathcal{O}(1)$ and $\alpha(G_n) \in \Omega(n^{1/2})$.*

Proof. Let G be a graph with $n = k^2$ vertices with $k \in \mathbb{Z}$ consisting of a $\sqrt{n}$-clique where each vertex of the clique except for one has additionally $\sqrt{n}$ pendants. Consequently G has $m < 3n$ edges and therefore $\langle \deg \rangle \in O(1)$. However its arborocity is $\alpha(G) \geq \lceil \frac{\sqrt{n}}{2} \rceil$ [2]. □

4 Structural Equivalence and Neighborhood Inclusion

The dominance preorder requires that dominating vertices are adjacent. This is a severe restriction, as all non-adjacent pairs are necessarily incomparable. A natural extension of the dominance preorder that softens this requirement is the *vicinal preorder* [5]. In the vicinal preorder, a vertex $w \in V$ *dominates* a vertex $v \in V$, $v \leq w$, if $N(v) \subseteq N[w]$. Another way to look at the vicinal preorder is that it is the union of the dominance preorder and the following.

In the *structural preorder* a vertex $w \in V$ dominates a vertex $v \in V$, $v \leq_n w$, if $N(v) \subseteq N(w)$. Analogously to the dominance preorder the subscript indicates that this relation can only exist between non-adjacent pairs of vertices. The resulting equivalence classes induce independent sets, and vertices that dominate each other are also known as *structurally equivalent* [12], or *false twins*. As the vicinal preorder is the union of dominance and structural preorder, each equivalence class induces either a clique or an independent set. The graphs for which the vicinal preorder is complete are known as *threshold graphs* [15].

Computing the set of false twins as well as recognizing threshold graphs can be done in $\mathcal{O}(m)$ time [8,11,18]. Moreover, constructing the dominance preorder using Algorithm 1 and counting cycles of length 4 (the problem underlying the structural and thus the vicinal preorder) is possible in time $\mathcal{O}(\alpha(G)m)$ [2]. However, these algorithms cannot be adapted for our purposes without increasing their running time. This is due to the fact that the sizes of structural and vicinal preorder are not bounded by $\mathcal{O}(\alpha(G)m)$, as we will show now using a concept closely related to the structural and vicinal preorder: *Subset partial orders*.

Given a family of subsets of a domain, the subset partial order represents all the subset inclusions between the subsets. Expressing our problem in terms of subset partial orders, the domain is the vertex set, the subsets are the neighborhoods of the vertices and the preorders correspond to the subset partial order.

Yellin and Jutla [23] constructed subset partial orders of size $\Theta(m^2/\log^2 m)$, which was later shown to be a tight upper bound [19]. The example below adapts Yellin and Jutla's construction to graphs. It shows the $\Omega(m^2/\log^2 m)$ lower bound for the structural and vicinal preorder, and demonstrates that we cannot hope to construct both preorders in time $\mathcal{O}(\alpha(G)m)$ like the dominance preorder.

Theorem 3. *There exists a family of graphs for which even the transitive reductions of both vicinal and structural preorder have size $\Theta(m^2/\log^2 m)$ and thus $\omega(\alpha(G)m)$.*

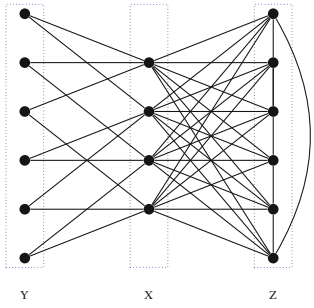

Fig. 1. Graph G_4 as produced by the construction in the proof of Theorem 3.

Proof. Let k denote an even positive integer. Let $X = \{x_1, \ldots, x_k\}$, $Y = \{y_1, \ldots, y_{\binom{k}{k/2}}\}$ and $Z = \{z_0, \ldots, z_{\binom{k}{k/2}-1}\}$ be disjoint sets. Let $A_1, \ldots, A_{\binom{k}{k/2}}$ be the subsets of X of size $k/2$. We construct the graph G_k as follows: The vertex set of G_k is given by $V = X \uplus Y \uplus Z$. Each vertex y_i is adjacent to all vertices in A_i. Finally, each vertex z_i is adjacent to all vertices in X, to vertex $z_{i-1 \mod \binom{k}{k/2}}$ and to $z_{i+1 \mod \binom{k}{k/2}}$; i.e., the vertices in Z form a cycle. Figure 1 exemplifies the construction for $k = 4$.

First, the graph has $m = \frac{k}{2}\binom{k}{k/2} + (k+1)\binom{k}{k/2} \in \Theta(k\binom{k}{k/2})$ edges. Second, note that all vertices in Z dominate all vertices in Y in the vicinal and structural preorder, but any two other vertices are incomparable. Thus, both preorders have size $\binom{k}{k/2}^2$, and since the preorders do not contain any transitive relationships, this is also the size of the transitive reductions. Finally, the graph has arboricity at most $k+2$, as we can cover all edges by k stars centered at the vertices in X and two paths that cover the edges of the cycle within Z.

Using Stirling's formula, we obtain $\binom{k}{k/2} \in \Theta(2^k/\sqrt{k})$ and thus $k \in \Theta(\log m)$. Putting it all together, the transitive reductions of the preorders have size $\binom{k}{k/2}^2 \in \Theta(m^2/\log^2 m) \subseteq \Omega(\alpha(G)m^2/\log^3 m) \subseteq \omega(\alpha(G)m)$. □

Several algorithms have been developed that compute the subset partial order in $\mathcal{O}(m^2/\log m)$ randomized or worst-case time [19,20,23]. However, these algorithms require substantive book keeping [19], cannot be generalized to weighted edges [20] or use complex data structures [23].

Algorithm 2 adapts a simple subset partial order algorithm introduced by Pritchard [20] to the vicinal preorder, and it can also be straightforwardly modified to determine the structural preorder instead. As this algorithm can also be used to count all cycles of length 4 in a graph, it can also be viewed as an adaption of Chiba and Nishizeki's algorithm `C4` [2].

Theorem 4. *Algorithm 2 determines the vicinal preorder of a simple undirected graph in time $\mathcal{O}(\Delta(G)m)$ with space linear in the size of the input.*

Proof. For each non-isolated vertex $v \in V$, the algorithm marks neighbors $u \in N(v)$ and vertices $w \in N(u) \setminus \{v\}$ at distance two. For a marked vertex w, $t[w]$

Algorithm 2: Vicinal Preorder

```
input  : graph G = (V, E)
output : partial ranking ≤ on V (neighborhood-inclusion)
initialize ≤ with v ≤ v for all v ∈ V;
for v ∈ V do
    for u ∈ N(v) do
        for w ∈ N[u] \ {v} do                    // use N(u) to determine ≤_n
            if w not marked with v then
                mark w with v;
                t[w] ← 0;
            increment t[w];
            if t[w] = deg(v) then add v ≤ w
```

holds the number of times it was encountered from a neighbor u of v (plus one for the neighbors themselves). This counter reaches $\deg(v)$ if and only if all (other) neighbors of v are also neighbors of w.

As for the running time, note that the first two loops yield $\sum_{v \in V} \deg(v) \in \mathcal{O}(m)$ iterations in total. During each iteration, the inner loop is executed $\deg(u) \leq \Delta(G)$ times. □

We note that Pritchard also gave an optimized variant of the algorithm that runs in $\mathcal{O}(\min\{m^2/\log n,\ \Delta(G)m\})$ time [20]. This can be a substantial improvement on sparse graphs with $o(n \log n)$ edges, and the algorithm can also be faster on graphs with $o(\log n)$ high-degree vertices. However, there appears to be no simple generalization of this optimization to weighted edges.

4.1 A Heuristic Based on Modular Decomposition

Closely related to the problem of computing the preorders is modular decomposition. The modular decomposition of a graph can be computed in $\mathcal{O}(m)$ time [7], and it lends itself to a heuristic approach to compute the dominance, structural and vicinal preorders on unweighted graphs that we will describe now.

In modular decomposition, a module defines a subset $M \subseteq V$ such that all vertices in M have exactly the same neighborhood in $V \setminus M$. A module is strong if there is no other module overlapping it. The modular decomposition tree $\mathrm{MD}(G)$ represents the inclusion structure of strong modules in the graph. The representative graph $R(M)$ of the module M is the quotient graph $G[M]/P$, where P is the partition of M given by the child modules of M in $\mathrm{MD}(G)$; in other words, it is the graph obtained from the subgraph $G[M]$ by contracting all child modules into single vertices. There are three types of strong modules: In a series module, $R(M)$ is a complete graph; in a parallel module, $R(M)$ is an empty graph; otherwise, the module is prime.

The heuristic computes the preorders by walking the modular decomposition tree $\mathrm{MD}(G)$. Suppose we are currently at module M in the walk of $\mathrm{MD}(G)$ and consider two vertices $v, w \in V$ that are contained in different child modules

C_v, C_w of M in $\mathrm{MD}(G)$. The heuristic decides the relation between v and w in the dominance and structural preorders as summarized in Table 1. The vicinal preorder arises by combining the cases for the other two preorders.

Table 1. Modular decomposition heuristic: ordering of v and w based on the type of the containing module M and the distinct children $C_v \ni v$, $C_w \ni w$ of M in $\mathrm{MD}(G)$

M	Dominance preorder	Structural preorder
Series	$N[v] \subseteq N[w] \iff C_w = \{w\}$	$N(v) \not\subseteq N(w)$
Parallel	$N[v] \not\subseteq N[w]$	$N(v) \subseteq N(w) \iff C_v = \{v\}$
Prime	$N[v] \subseteq N[w] \iff$ 1. $N_{R(M)}[C_v] \subseteq N_{R(M)}[C_w]$ 2. $C_w = \{w\}$ or C_w is series with child $\{w\}$	$N(v) \subseteq N(w) \iff$ 1. $N_{R(M)}(C_v) \subseteq N_{R(M)}(C_w)$ 2. $C_v = \{v\}$ or C_v is parallel with child $\{v\}$

The bottleneck in this heuristic is condition 1 for prime modules. While we can construct the representative graphs in $\mathcal{O}(m)$ time [10], we also need algorithms that compute the preorders on them. Using Algorithms 1 and 2, the heuristic computes the dominance preorder in time $\mathcal{O}(m+\alpha'(G)m') \subseteq \mathcal{O}(\alpha(G)m)$ and the structural and vicinal preorders in time $\mathcal{O}(m+\Delta'(G)m'+|\leq|) \subseteq \mathcal{O}(\Delta(G)m)$, where $\alpha'(G)$ and $\Delta'(G)$ denote the maximum arboricity and the maximum degree of a representative prime graph in G, m' is the total number of edges in representative prime graphs in G, and $|\leq|$ refers to the size of the output. This heuristic can thus significantly improve runtime on decomposable graphs.

5 Positional Dominance

The notions of dominance considered so far all require adjacency with identical neighbors. Common centrality indices, on the other hand, are typically invariant under automorphisms. In degree centrality, for instance, it is sufficient to have more neighbors, no matter which. Positional dominance [1] generalizes such assumptions by allowing comparison of neighbors using sets of admissible permutations. We here consider the case in which any neighbor with at least the same vertex weight and at least an equally strong relationship may serve as a replacement.

A vertex w dominates vertex v w.r.t. positional dominance,

$$v \leq w \quad \text{if} \quad \begin{cases} \text{there ex. } \pi : V \to V \text{ such that } \omega(u) \leq \omega(\pi(u)) \text{ and} \\ \omega((v,u)) \leq \omega((w,\pi(u))) \quad \forall (v,u) \in E : \omega((v,u)) \neq 0\,. \end{cases}$$

Restricting π to the identity permutation or transpositions we can derive the structural and dominance preorders, therefore positional dominance is a generalization of the previous notions. Note that diagonal entries and the dyads $(v,w),(w,v)$ may be treated specially when comparing v and w in the positional dominance approach, however we will not address this topic in more detail here.

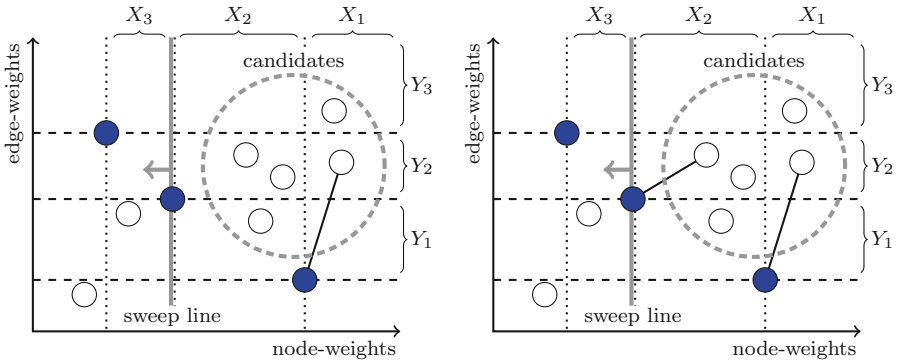

Fig. 2. Sweep line approach: blue/white vertices are neighbors of v/w, and white vertices are partitioned into bins X_i and Y_j along dotted and dashed lines, respectively. Vertices connected by a black line have been matched. (left) potential candidates for the matching of the vertex under the sweep line; (right) matching an unmatched vertex in the non-empty bin Y_k (here Y_2) that contains the vertices with smallest edge-weight greater than or equal to the one of the vertex under the sweep line. (Color figure online)

The straightforward approach to decide whether $v \leq w$ or not consists of two phases.[1] In the first phase for each vertex $u \in N(w)$ the subset of vertices in $N(v)$ that is dominated by u is computed, which requires $\mathcal{O}(k + \deg(w) \log \deg(v))$ time [14] with k denoting the total number of dominance pairs. Based on these dominance relations, it is tested in the second phase if there exists a mapping π such that $v \leq w$. Finding this mapping is equivalent to finding a perfect matching in the bipartite graph induced by the dominance relations and can be done in $\mathcal{O}(\sqrt{\deg(v) + \deg(w)}k)$ [9]. Consequently computing the positional dominance preorder using this approach has a complexity of $\mathcal{O}(nm\Delta(G)^{3/2})$.

In the following we will show that this problem can be solved more efficiently using a greedy sweep line approach, cf. Algorithm 3 and Fig. 2. The basic idea of this approach is to process all neighbors v_i of vertex v in decreasing order of their vertex-weights. The algorithm maintains bins $\mathcal{Y} = (Y_1, \ldots, Y_{\deg(v)})$ that partition the set of unmatched candidates u with vertex-weight $\omega(u) \geq \omega(v_i)$. Here, a bin Y_j contains all those unmatched candidates u whose associated edge-weights lie between $\omega(v, v_{\sigma(j)}) \leq \omega(w, u) < \omega(v, v_{\sigma(j+1)})$, where σ is a permutation such that neighbors $v_{\sigma(1)}, \ldots, v_{\sigma(\deg(v))}$ of v are sorted in increasing order of their associated edge-weights. Hence, the bins $Y_{\sigma^{-1}(i)}, \ldots, Y_{\deg(v)}$ contain all unmatched candidates with edge-weight at least $\omega(v, v_i)$. To find a vertex to match with v_i, we thus identify a non-empty bin Y_k with $k \geq \sigma^{-1}(i)$. If there are several such bins, we choose the one with smallest index $k \geq \sigma^{-1}(i)$. We then match v_i with an arbitrary vertex in bin Y_k. This greedy matching is correct since v's remaining unprocessed neighbors all have smaller vertex-weights than all the vertices in the bins $\mathcal{Y}$ due to the way we process the neighbors of v. Matching v_i with a candidate from the non-empty bin Y_k with minimal index $k \geq \sigma^{-1}(i)$,

[1] In the following we assume that $\deg(v) \leq \deg(w)$, since otherwise w cannot dominate v w.r.t. positional dominance.

Algorithm 3: Positional Dominance Test $v \leq w$

input : graph $G = (V, E; \omega : V \cup E \to \mathbb{R})$ and $v, w \in V$
data : bins $\mathcal{X} = (X_1, \ldots, X_{\deg(v)}), \mathcal{Y} = (Y_1, \ldots, Y_{\deg(v)})$,
vertex array of bin indices $b[u]$, list of indices of non-empty bins T
output : boolean indicating whether $v \leq w$, or not

if $\deg(w) < \deg(v)$ **then return** *FALSE*
let $N(v) = \langle v_1, \ldots, v_{\deg(v)} \rangle$ s.t. $\omega(v_1) \geq \cdots \geq \omega(v_{\deg(v)})$;
let σ be permutation s.t. $\omega(v, v_{\sigma(1)}) \leq \cdots \leq \omega(v, v_{\sigma(\deg(v))})$;
partition $N(w)$ into $X_i = \{u \in N(w) : \omega(v_i) \leq \omega(u) < \omega(v_{i-1})\}$ where $\omega(v_0) := \infty$;
for $u \in N(w)$ **do**
  if $\omega(v, v_{\sigma(1)}) \leq \omega(w, u)$ **then** $b[u] \leftarrow \max\{k : \omega(v, v_{\sigma(k)}) \leq \omega(w, u)\}$
$T, Y_1, \ldots, Y_m \leftarrow \emptyset$;
for $i = 1, \ldots, \deg(v)$ **do**
  for $u \in X_i$ **do**
    if $\omega(v, v_{\sigma(1)}) \leq \omega(w, u)$ **then**
      if $Y_{b[u]} = \emptyset$ **then** $T \leftarrow T \cup \{b[u]\}$
      $Y_{b[u]} \leftarrow Y_{b[u]} \cup \{u\}$
  if $k < \sigma^{-1}(i)$ *for all* $k \in T$ **then return** *FALSE*
  $k \leftarrow \min\{\ell \in T : \ell \geq \sigma^{-1}(i)\}$;
  remove some vertex u from Y_k `// match` v_i `and some vertex` $u \in Y_k$
  if $Y_k = \emptyset$ **then** $T \leftarrow T \setminus \{k\}$
return *TRUE*

consequently, retains those candidates in the bins that have the highest potential to also dominate the remaining neighbors of v respective their edge-weights.

Before we actually prove the correctness and running time of this algorithm, we first show an invariant of the outer for loop. Assume that vertex v is dominated by vertex w w.r.t. positional dominance via the permutation $\pi' : V \to V$. This means that Algorithm 3 would succeed if it matched according to the permutation π', since neighbor $\pi'(v_i)$ of w would always be available for matching in some bin while processing v_i. Now let $\mathcal{Y} = (Y_1, \ldots, Y_{\deg(v)})$ be the actual bins produced by the algorithm, and $\mathcal{Y}' = (Y'_1, \ldots, Y'_{\deg(v)})$ be the bins that would be produced by the algorithm if it matched according to the permutation π' instead. Observe that at any point during the execution of the algorithm, we have $\sum_{j=1}^{\deg(v)} |Y_j| = \sum_{j=1}^{\deg(v)} |Y'_j|$, and T and T' always contain the indices of all non-empty bins $\mathcal{Y}$ and $\mathcal{Y}'$, respectively. We say that $\mathcal{Y}$ *covers* $\mathcal{Y}'$ if and only if for all $k \in \{1, \ldots, \deg(v)\}$ we have $\sum_{j=k}^{\deg(v)} |Y_j| \geq \sum_{j=k}^{\deg(v)} |Y'_j|$.

Lemma 1. *$\mathcal{Y}$ covers $\mathcal{Y}'$ at the end of each successful iteration performed by the outer for loop.*

Proof. We prove the loop invariant by induction on the number of loop iterations performed by the outer for loop.

– Basis: Before the first iteration of the outer for loop, all bins are empty, so $\mathcal{Y}$ trivially covers $\mathcal{Y}'$.

– Inductive step: By induction, $\mathcal{Y}$ covers $\mathcal{Y}'$ at the end of the $(i-1)$-th iteration. During the i-th iteration, the inner for loop adds the same vertices to the bins in $\mathcal{Y}$ and $\mathcal{Y}'$, therefore at the end of this loop $\mathcal{Y}$ still covers $\mathcal{Y}'$.
Finally, Algorithm 3 tries to match the current neighbor v_i of v with some neighbor of w. Since w dominates v via permutation π', $\omega(w, \pi'(v_i))$ is at least as large as $\omega(v, v_i)$, and since $\pi'(v_i)$ thus must be matchable, $\pi'(v_i)$ must be in some bin Y'_k. Additionally, $\mathcal{Y}$ still covers $\mathcal{Y}'$ after the inner for loop, so there is a non-empty bin Y_ℓ with $\sigma^{-1}(i) \leq \ell$ minimal. Algorithm 3 will pick a vertex from Y_ℓ and match it with v_i. If $\ell \leq k$, then the invariant still holds trivially after the matching. If $\ell > k$, observe that $\sum_{j=\ell}^{\deg(v)} |Y_j| = \sum_{j=k}^{\deg(v)} |Y_j| \geq \sum_{j=k}^{\deg(v)} |Y'_j| \geq 1 + \sum_{j=k+1}^{\deg(v)} |Y'_j|$ before the removal, as bins $Y_k, \ldots, Y_{\ell-1}$ are empty. Thus, $\mathcal{Y}$ still covers $\mathcal{Y}'$ after removing vertices from Y_ℓ and Y'_k in both cases. □

Theorem 5. *Algorithm 3 decides for a given pair of vertices v, w with $\deg(v) \leq \deg(w)$ and weights ω in $\mathcal{O}(\deg(w) \log \deg(v))$ time whether $v \leq w$ or not.*

Proof. We will start by proving the correctness of the algorithm and thereafter show the correctness of the claimed running time.

Correctness: Assume for now that w dominates v w.r.t. positional dominance via permutation π'. Suppose that Algorithm 3 already succeeded in performing $0 \leq i - 1 < \deg(v)$ iterations of the outer for loop, and let $\mathcal{Y}$ and $\mathcal{Y}'$ denote the respective bins at the beginning of the i-th iteration. During the i-th iteration, Algorithm 3 tries to match neighbor v_i of v with some neighbor of w. In π', v_i has already been matched with $\pi'(v_i) \in N(w)$. First, assume that $\pi'(v_i)$ was already in a bin Y'_k at the beginning of the i-th iteration; i.e., it was already a candidate in previous iterations. Since $\mathcal{Y}$ covers $\mathcal{Y}'$ (Lemma 1), there is a non-empty bin Y_ℓ with $\ell \geq k$. Since $\ell \geq k \geq \sigma^{-1}(i)$, we have $\omega(w, u) \geq \omega(v, v_i)$ for any vertex $u \in Y_\ell$, so the algorithm finds a vertex to match with v_i in the i-th iteration. Next, suppose that $\pi'(v_i)$ was newly added to the bins $\mathcal{Y}'$ in the inner for loop of the i-th iteration. All newly added vertices are inherently unmatched in the algorithm and thus can be matched with v_i. Therefore, the algorithm will also succeed in performing the i-th iteration, and hence, correctly decide that $v \leq w$.

Conversely, assume the algorithm succeeds. Let π be the permutation computed during the run of the algorithm. The algorithm ensures that any neighbor v_i of v is only matched with a single neighbor $\pi(v_i)$ of w if $\omega(\pi(v_i)) \geq \omega(v_i)$, as $\pi(v_i)$ will not be added to the bins $\mathcal{Y}$ otherwise, and $\omega(w, \pi(v_i)) \geq \omega(v, v_i)$, as otherwise $\pi(v_i)$ cannot be extracted from a bin Y_k, $k \geq \sigma^{-1}(i)$. Thus, w dominates v w.r.t. positional dominance via permutation π. □

Time complexity: Sorting the neighbors of v (lines 2–3) and (pre-)binning the neighbors of w (lines 4–6) can be done in $\mathcal{O}(\deg(w) \log \deg(v))$ time. It remains to be shown that the work done by the outer for loop does not exceed this complexity. The cost of the outer for loop is composed of (i) adding neighbors of w to a bin Y_k, (ii) adding indices of non-empty bins to T (iii) testing if T contains a bin Y_k with $k \geq \sigma^{-1}(i)$ and (iv) possibly removing the index of a bin from T if that bin becomes empty again. Since each neighbor of w is sorted into a

bin Y_k at most once, (i) costs in total $\mathcal{O}(\deg(w))$ time. Furthermore, there are at most $\deg(v)$ deletions and tests in T, so steps (ii), (iii) and (iv) are performed at most $\mathcal{O}(\deg(v))$ times. When T is implemented by a specialiced data structure on a bounded integer domain like a van Emde Boas tree [3], (ii), (iii) and (iv) cost in total $\mathcal{O}(\deg(v) \log\log \deg(v))$ time. Thus, the total running time of the outer for loop of Algorithm 3 is in $\mathcal{O}(\deg(w) + \deg(v) \log\log \deg(v))$. □

To compute the positional dominance preorder, we can sort the neighborhoods of all vertices in advance in $\mathcal{O}(m \log \Delta(G))$ time. Then we no longer need to sort the neighborhoods (lines 2–3) and (pre-)binning (lines 4–6) is possible in $\mathcal{O}(\deg(w))$ time in Algorithm 3. That means that on pre-sorted neighborhoods, the runtime is dominated by the outer for loop, which requires $\mathcal{O}(\deg(w) + \deg(v) \log\log \deg(v))$ time. Hence, we can compute the positional dominance preorder of a graph G with weights ω in time $\mathcal{O}(nm \log\log \Delta(G))$, since

$$\sum_{\substack{v,w \in V \\ \deg(v) \leq \deg(w)}} (\deg(w) + \deg(v) \log\log \deg(v)) \leq 2n \left(\sum_{v \in V} \deg(v) \right) \log\log \Delta(G)$$
$$= 4nm \log\log \Delta(G).$$

6 Conclusion

We studied various notions of dominance which can serve as potential building blocks for the generalization of the concept of centrality. Using a greedy sweep line approach, cf. Algorithm 3, we were able to show that positional dominance can be computed in $\mathcal{O}(nm \log\log \Delta(G))$ time compared to $\mathcal{O}(nm\Delta(G)^{3/2})$ required by a straight-forward algorithm to solve this problem. For this problem, we see the greatest potential for further runtime improvements in avoiding some of the pairwise comparisons between vertices.

Restricting positional dominance to the identity permutation, i.e., assuming heterogeneity, translates into the structural preorder, which is a restriction of the vicinal preorder and a variant of (vertex) dominance. With Algorithm 1 we presented an algorithm running in $\mathcal{O}(a(G)m)$ to compute the dominance preorder. The running time may be far from linear in the size of input and output, for social networks, however, where the arboricity is often negligible [4], it is acceptable, not least since we are not aware of any faster solution to this problem. While the main challenge in the computation of the structural preorder lies in finding cycles of length four we proved that a running time of $\mathcal{O}(a(G)m)$, which is the running time of an efficient algorithm to solve this problem [2], is not achievable as the size of the preorder can already be much larger. As a result of this finding we proposed with Algorithm 2 a procedure to compute the structural as well as vicinal preorder in time $\mathcal{O}(\Delta(G)m)$. For computing dominance, structural and vicinal preorder, we additionally presented a heuristic that can yield substantial speed-ups on unweighted graphs that are decomposable through modular decomposition.

References

1. Brandes, U.: Network positions. Methodol. Innov. **9**, 2059799116630650 (2016)
2. Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM J. Comput. **14**(1), 210–223 (1985)
3. van Emde Boas, P.: Preserving order in a forest in less than logarithmic time and linear space. Inf. Process. Lett. **6**(3), 80–82 (1977)
4. Eppstein, D., Spiro, E.S.: The h-index of a graph and its application to dynamic subgraph statistics. J. Graph Algorithms Appl. **16**(2), 543–567 (2012)
5. Foldes, S., Hammer, P.L.: The Dilworth number of a graph. Ann. Discret. Math. **2**, 211–219 (1978)
6. Freeman, L.C.: A set of measures of centrality based on betweenness. Sociometry **40**(1), 35–41 (1977)
7. Habib, M., Paul, C.: A survey of the algorithmic aspects of modular decomposition. Comput. Sci. Rev. **4**(1), 41–59 (2010)
8. Heggernes, P., Kratsch, D.: Linear-time certifying recognition algorithms and forbidden induced subgraphs. Nordic J. Comput. **14**(1–2), 87–108 (2007)
9. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. **2**(4), 225–231 (1973)
10. Lagraa, S., Seba, H.: An efficient exact algorithm for triangle listing in large graphs. Data Min. Knowl. Disc. **30**(5), 1350–1369 (2016)
11. Lerner, J.: Role assignments. In: Brandes, U., Erlebach, T. (eds.) Network Analysis. LNCS, vol. 3418, pp. 216–252. Springer, Heidelberg (2005). doi:10.1007/978-3-540-31955-9_9
12. Lorrain, F., White, H.C.: Structural equivalence of individuals in social networks. J. Math. Soc. **1**(1), 49–80 (1971)
13. Lü, L., Chen, D., Ren, X.L., Zhang, Q.M., Zhang, Y.C., Zhou, T.: Vital nodes identification in complex networks. Phys. Rep. **650**, 1–63 (2016)
14. Lueker, G.S.: A data structure for orthogonal range queries. In: 19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16–18 October 1978, pp. 28–34 (1978)
15. Mahadev, N.V., Peled, U.N.: Threshold Graphs and Related Topics, Annals of Discrete Mathematics, vol. 56. Elsevier, Amsterdam (1995)
16. Nash-Williams, C.S.J.A.: Decomposition of finite graphs into forests. J. Lond. Math. Soc. **39**(1), 12 (1964)
17. Ortmann, M., Brandes, U.: Triangle listing algorithms: back from the diversion. In: Proceedings of the 16th Workshop on Algorithm Engineering and Experiments (ALENEX 2014), pp. 1–8 (2014)
18. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Comput. **16**(6), 973–989 (1987)
19. Pritchard, P.: On computing the subset graph of a collection of sets. J. Algorithms **33**(2), 187–203 (1999)
20. Pritchard, P.: A simple sub-quadratic algorithm for computing the subset partial order. Inf. Process. Lett. **56**(6), 337–341 (1995)
21. Sabidussi, G.: The centrality index of a graph. Psychometrika **31**(4), 581–603 (1966)
22. Schoch, D., Brandes, U.: Re-conceptualizing centrality in social networks. Eur. J. Appl. Math. **27**(6), 971–985 (2016)
23. Yellin, D.M., Jutla, C.S.: Finding extremal sets in less than quadratic time. Inf. Process. Lett. **48**(1), 29–34 (1993)