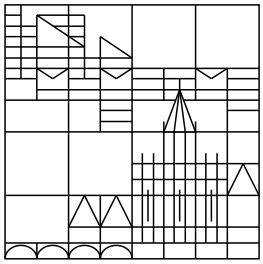
Universität Konstanz



Edge-Disjoint Paths in Planar Graphs with Short Total Length

Ulrik Brandes
Gabriele Neyer
Dorothea Wagner

Konstanzer Schriften in Mathematik und Informatik
Nr. 19, Dezember 1996

ISSN 1430–3558

Universität Konstanz
Postfach 5560/D 188, 78434 Konstanz, Germany
Email: preprints@informatik.uni–konstanz.de
WWW: http://www.informatik.uni–konstanz.de/Schriften

Edge–Disjoint Paths in Planar Graphs with Short Total Length*

Ulrik Brandes† Gabriele Neyer‡ Dorothea Wagner§

19/1996
Fakultät für Mathematik und Informatik
Universität Konstanz

Abstract

The problem of finding edge–disjoint paths in a planar graph such that each path connects two specified vertices on the outer face of the graph is well studied. The "classical" Eulerian case introduced by Okamura and Seymour [7] is solvable in linear time [10]. So far, the length of the paths were not considered. In this paper now, we prove that the problem of finding edge–disjoint paths of minimum total length in a planar graph is $\mathcal{NP}$–hard, even if the graph fullfills the Eulerian condition and the maximum degree is four. Minimizing the length of the longest path is $\mathcal{NP}$–hard as well. Efficient heuristics based on the algorithm from [10] are presented that determine edge–disjoint paths of small total length. We have implemented these heuristics and have studied their behaviour. It turns out that some of the heuristics are empirically very successful.

1 Introduction

Let $G = (V, E)$ be simple, undirected, planar graph given along with a fixed combinatorial embedding, that is, the adjacency list of each vertex is sorted according to a fixed geometric embedding in the plane, and there is one designated face, the *outer face*. Consider a set $N = \{\{s_1, t_1\}, \ldots, \{s_k, t_k\}\}$, where $s_1, t_1, \ldots, s_k, t_k$ are vertices of G. The elements of N are called *nets* and the s_i, t_i are called *terminals*. Notice, that the terminals are not necessarily distinct. A graph $G = (V, E)$ together with a set of nets $N = \{\{s_1, t_1\}, \ldots, \{s_k, t_k\}\}$ satisfies the *Eulerian condition* if and only if the graph $(V, E + \{s_1, t_1\} + \cdots + \{s_k, t_k\})$ is Eulerian. The problem is to determine edge–disjoint paths $p_1, \ldots, p_k$, such that p_i connects s_i with t_i for $i = 1, \ldots, k$. The basic result due to Okamura and Seymour is a theorem that gives a necessary and sufficient condition for solvability. Efficient algorithms based on the proof of this theorem are given in [5, 1, 4]. An algorithm solving the problem in linear time is presented in [10]. The complexity status is open for the case that the Eulerian condition is dropped.

So far, the length of the paths were only considered for very restricted cases, i. e., where the graph is a rectilinear grid [11], [2, 9]. The only case known, where edge–disjoint paths of minimum total length can be determined in polynomial time is when the instance is a dense channel [2]. In this case, the problem is even solvable in time linear in the number of nets [9]. For convex grids, an efficient heuristic to determine edge–disjoint paths of small total length is presented in [11]. In this paper now, we prove that finding edge–disjoint paths of minimum total length in planar

*The authors acknowledge the *Deutsche Forschungsgemeinschaft* for supporting this research under grant *Wa 654/10-2*.

†Fakultät für Mathematik und Informatik, Universität Konstanz, Universitätsstraße 10, 78434 Konstanz, Germany, ulrik.brandes@uni-konstanz.de

‡Theoretical Computer Science, ETH Zentrum, ETH Zürich, CH-8092 Zürich, Switzerland, neyer@inf.ethz.ch

§Fakultät für Mathematik und Informatik, Universität Konstanz, Universitätsstraße 10, 78434 Konstanz, Germany, dorothea.wagner@uni-konstanz.de. Communicate with third author.

Eulerian instances of maximum degree four is $\mathcal{NP}$-hard, and minimizing the length of the longest path is $\mathcal{NP}$-hard as well. We present efficient heuristics based on the algorithm from [10] that determine edge–disjoint paths of small total length. The heuristics have been implemented and included in the algorithm software package PlaNet [6]. They are studied empirically, and for not too large instances the results are compared to the solutions determined by an exact approach[1] with exponential time complexity.

This paper is organized as follows. After some preliminaries introduced in Section 2, we give the $\mathcal{NP}$–hardness proofs in Section 3. In Section 4 we describe the heuristics and present an experimental study of their empirical behaviour.

2 Preliminaries

Problem 2.1 Edge–Disjoint Paths Problem

Given: *An instance (G, N) consisting of a planar graph $G = (V, E)$ and a set of terminal pairs $N = \{\{s_1, t_1\}, \ldots, \{s_k, t_k\}\}$. The graph G is embedded in the plane such that the terminals $s_1, \ldots, s_k, t_1, \ldots, t_k$ lie on the boundary of the outer face. (G, N) satisfies the Eulerian condition.*

Problem: *Find k edge–disjoint paths in G connecting s_i and t_i, for $1 \leq i \leq k$.*

In the following, we assume G to be biconnected. For a non–biconnected graph the problem can be easily solved by considering its biconnected components separately. We first introduce the algorithm from [10] which solves the edge–disjoint paths problem in linear time. For technical reasons, G is modified such that all terminals have degree 1 and all other vertices have even degree. Obviously, an instance can easily be transformed into a completely equivalent instance that fulfills this assumption. Now, let x be an arbitrary terminal, called *start terminal*. W.l.o.g., according to a counterclockwise ordering of all terminals starting with x, s_i precedes t_i for $i = 1, \ldots, k$, and t_i precedes t_{i+1} for $i = 1, \ldots, k-1$. The latter clearly means that in a sense all t–terminals are sorted in increasing order. The algorithm is based on "right–first search", i. e., a depth–first search where in each search step the edges are searched from right to left. It consists of two phases. In the first phase, an "easier" instance $(G, N^{()})$ of *parenthesis structure* is solved. Then in the second phase, a solution to the instance (G, N) is determined based on the solution for $(G, N^{()})$.

For the first phase, consider the $2k$–string of s-terminals and t-terminals on the outer face in counterclockwise ordering, starting with x. The i^{th} terminal is assigned a *left parenthesis* if it is an s–terminal, and a *right parenthesis* otherwise. The resulting $2k$–string of parentheses is then a string of left and right parentheses that can be paired correctly, i. e., such that the pairs of parentheses are properly nested. The terminals are now newly paired according to this (unique) correct pairing of parentheses, i. e., an s–terminal and a t-terminal are paired if and only if the corresponding parentheses match. It is easy to see that $(G, N^{()})$ is solvable, if (G, N) is. Procedure 2.2 determines such a solution $(q_1, \ldots, q_k)$ for $(G, N^{()})$. This solution will be used for the determination of the final solution. In contrast to the original nets, we denote the nets in $N^{()}$ by $\{s_1^{()}, t_1^{()}\}, \ldots, \{s_k^{()}, t_k^{()}\}$, and we assume w. l. o. g. that $t_i = t_i^{()}$ for $i = 1, \ldots, k$. The paths q_i are determined by a right–first search. Let $v \in V$, and let e be incident to v. We will say that the *next free edge after* e with respect to v is the first free edge to follow e in the adjacency list of v in counterclockwise ordering.

Procedure 2.2

for $i := 1$ **to** k **do**
  let q_i initially consist of the unique edge incident to $s_i^{()}$;
  $v :=$ the unique vertex adjacent to $s_i^{()}$;

[1]The authors want to thank Martin Oellrich and Andreas S. Schulz for making the implementation of an exact method using CPLEX available.

while v is no terminal **do**
let $\{v, w\}$ be the next free edge after the leading edge of q_i with respect to v;
add $\{v, w\}$ to q_i;
$v := w$;
if $v \neq t_i^{()}$ **then** stop: **return** "unsolvable";
return $(q_1, \ldots, q_k)$;

The *auxiliary paths* $q_1, \ldots, q_k$ yield a directed *auxiliary graph* $A(G, N, x)$ of instance (G, N) w. r. t. start terminal x. Just orient all edges on the paths $q_1, \ldots, q_k$ according to the direction in which they are traversed during the procedure. Then $A(G, N, x)$ consists of all vertices of G and of all oriented edges. The solution $p_1, \ldots, p_k$ for the original instance (G, N) is now determined in the auxiliary graph. That is, edges that are not contained in the auxiliary graph will not be occupied by a path $p_1, \ldots, p_k$ of the final solution. Even more, the edges occupied by the final solution are exactly the edges of the auxiliary graph. The solution paths p_i are determined by a "directed" right–first search. That is, edges that belong to $A(G, N, x)$ are used according to their orientations in $A(G, N, x)$.

Algorithm 2.3

determine $A(G, \mathcal{N}, x)$ for an arbitrary start terminal x;
for $i := 1$ **to** k **do**
let p_i initially consist of the unique edge leaving s_i in $A(G, \mathcal{N}, x)$;
$v :=$ the head of this edge;
while v is no terminal **do**
let (v, w) be the next free edge leaving v after the leading edge of p_i with respect to v;
add (v, w) to p_i;
$v := w$;
if $v \neq t_i$ **then** stop: **return** "unsolvable";
return $(p_1, \ldots, p_k)$;

Algorithm 2.3 can be implemented to run in linear time using a special case of Union–Find [3]. For a proof of correctness see [10]. We will focus on the following optimization problem.

Problem 2.4 Minimum Edge–Disjoint Paths Problem

Instance: *An instance (G, N) consisting of a planar graph $G = (V, E)$ and a set of terminal pairs $N = \{\{s_1, t_1\}, \ldots, \{s_k, t_k\}\}$ which satisfies the Eulerian condition. The graph G is embedded in the plane such that $s_1, \ldots, s_k, t_1, \ldots, t_k$ lie on the boundary of the outer face.*

Problem: *Find k edge–disjoint paths $p_1, \ldots, p_k$ in G connecting s_i and t_i, for $1 \leq i \leq k$, such that $\sum_{i=1}^{k} length(p_i)$ is minimum. (The length of a path p is the number of edges on p.)*

3 $\mathcal{NP}$–Completeness Proof

We prove that the decision problem corresponding to Problem 2.4 is $\mathcal{NP}$–complete, even if the maximum degree of the graph is four.

Problem 3.1 Minimum Edge–Disjoint Paths Decision Problem

Instance: *An integer K and an instance (G, N) consisting of a planar graph $G = (V, E)$ and a set of terminal pairs $N = \{\{s_1, t_1\}, \ldots, \{s_k, t_k\}\}$ which satisfies the Eulerian condition. The graph G is embedded in the plane such that $s_1, \ldots, s_k, t_1, \ldots, t_k$ lie on the boundary of the outer face.*

Question: *Exist edge–disjoint paths $p_1, \ldots, p_k$ in G connecting s_i and t_i, for $1 \leq i \leq k$, such that $\sum_{i=1}^{k} length(p_i) \leq K$?*

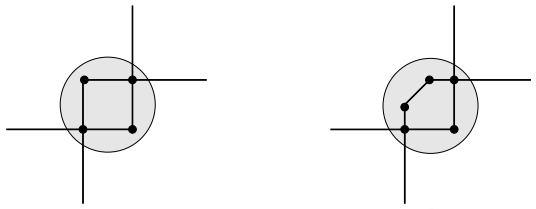


Figure 1: a) Super–vertex of type 1. b) Super–vertex of type 0.

Theorem 3.2 *The minimum edge–disjoint paths decision problem stated as Problem 3.1 is $\mathcal{NP}$–complete, even if the maximum degree of G is four.*

Proof It is easy to see that Problem 3.1 is in $\mathcal{NP}$. For the completeness proof we use a transformation from $3SAT$. An instance of $3SAT$ is given by a set U of variables, $|U| = n$, and a set C of clauses over U, $|C| = m$, such that $|c| = 3$ for $c \in C$. The problem is to decide if there is a satisfying truth assignment for C. We can assume w.l.o.g. that n is an even number. Since, if n is odd we can obviously add a "dummy variable".

Let $K := n(9m+1) + m(6n+1)$. We construct an even instance (G, N) consisting of a planar graph G with maximum degree four, and a set of nets whose terminals lie on the boundary of the outer face of G. G is a grid–like graph consisting of $2n$ horizontal lines i_1, i_2, $i = 1, \ldots, n$ and $3m$ vertical lines j_1, j_2, j_3, $j = 1, \ldots, m$. A horizontal line $i_l, l \in \{1, 2\}$ and a vertical line $j_r, r \in \{1, 2, 3\}$ meet in a "super–vertex" v_{i_l, j_r} of *type* 1 or *type* 0. A super–vertex of *type* 1 consists of four vertices and a super–vertex of *type* 0 consists of five vertices. See Figure 1. Variable $u_i \in U$ corresponds to two subsequent horizontal lines i_1, i_2, where u_i corresponds to i_1 and $\neg u_i$ corresponds to i_2. The clause $c_j \in C$ corresponds to the vertical lines j_1, j_2, j_3, where j_r corresponds to the r^{th} literal in c_j. For every variable $u_i \in U$ and every clause $c_j \in C$ we have a net $\{s_{u_i}, t_{u_i}\}$ respectively $\{s_{c_j}, t_{c_j}\}$. Terminal s_{u_i} lies on a vertex connected to the two leftmost super–vertices on horizontal lines i_1 and i_2, while terminal t_{u_i} lies on a vertex connected to the two rightmost super-vertices on horizontal lines i_1 and i_2. Analogously, terminal s_{c_j} lies on a vertex connected to the three uppermost super–vertices on vertical lines j_1, j_2 and j_3, and terminal t_{c_j} lies on a vertex connected to the three lowermost super–vertices on vertical lines j_1, j_2 and j_3. In order to guarantee that the instance is Eulerian, vertices corresponding to s_{u_i} respectively t_{u_i} are pairwise connected by an edge, i. e., edges $\{s_{u_i}, s_{u_{i+1}}\}$ and $\{t_{u_i}, t_{u_{i+1}}\}$ are added for $i = 1, \ldots, k-1$. Observe, that a shortest path connecting s_{u_i} and t_{u_i} resp. s_{c_j} and t_{c_j} has length $9m+1$ resp. $6n+1$. See Figure 2.

For super–vertex v_{i_l, j_r}, where horizontal line $i_l, l \in \{1, 2\}$ and vertical line $j_r, r \in \{1, 2, 3\}$ meet, we have

$$type(v_{i_l, j_r}) := \begin{cases} 0 & \text{if } l = 1 \text{ and } \neg u_i \text{ occurs in clause } c_j \text{ as } r^{th} \text{ literal} \\ & \text{or } l = 2 \text{ and } u_i \text{ occurs in clause } c_j \text{ as } r^{th} \text{ literal;} \\ 1 & \text{otherwise.} \end{cases}$$

By definition, a super–vertex v_{i_l, j_r} is of type 0 if and only if setting the corresponding literal $\neg u_i$ resp. u_i to false does not satisfy the corresponding clause c_j. Obviously, the two paths connecting s_{u_i} and t_{u_i} respectively s_{c_j} and t_{c_j} can be both shortest only if they meet in a super–vertex of type 1. See Figure 3 for an example.

Now, a satisfying truth assignment for an instance of $3SAT$ induces a solution to the corresponding instance of Problem 3.1 as follows. Net $\{s_{u_i}, t_{u_i}\}$ is routed along horizontal line i_1 if and only if u_i is set *true*. Net $\{s_{c_j}, t_{c_j}\}$ is routed along the leftmost vertical line $j_r, r \in \{1, 2, 3\}$ corresponding to a literal satisfying c_j. Then nets $\{s_{u_i}, t_{u_i}\}$ and $\{s_{c_j}, t_{c_j}\}$ meet only in super–vertices of type 1. Consequently, the total length of the solution is K.

On the other hand, in a solution of length at most K to the instance of Problem 3.1 any net $\{s_{u_i}, t_{u_i}\}$ has length $9m+1$ and any net $\{s_{c_j}, t_{c_j}\}$ has length $6n+1$. Therefore, nets only meet at

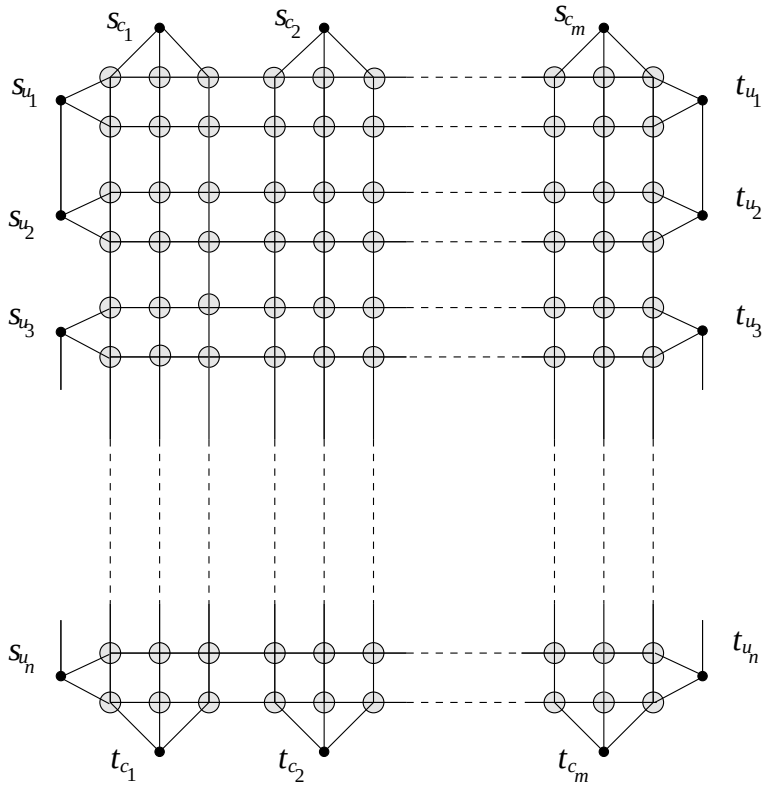


Figure 2: Generic example of an instance of Problem 3.1 corresponding to an instance of $3SAT$.

super–vertices of type 1. This induces a truth assignment for the instance of $3SAT$ where u_i is set true if and only if $\{s_{u_i}, t_{u_i}\}$ is routed along horizontal line i_1. If for a clause c_j the corresponding net is routed along horizontal line j_r then the net corresponding to the r^{th} literal of c_j passes j_r through a super–vertex of type 1. Consequently, this literal must be true and c_j is satisfied. □

Problem 3.3 Minimum Longest Path Problem

Instance: *An integer K and an instance (G, N) consisting of a planar graph $G = (V, E)$ and a set of terminal pairs $N = \{\{s_1, t_1\}, \ldots, \{s_k, t_k\}\}$ which satisfies the Eulerian condition. The graph G is embedded in the plane such that $s_1, \ldots, s_k, t_1, \ldots, t_k$ lie on the boundary of the outer face.*

Question: *Exist edge–disjoint paths $p_1, \ldots, p_k$ in G connecting s_i and t_i, for $1 \leq i \leq k$, such that the length of the longest path p_i is at most K?*

Corollary 3.4 *Problem 3.3 is $\mathcal{NP}$–complete, even if the maximum degree of G is four.*

Proof (Sketch) We use a slight modification of the reduction from Theorem 3.2. The instance for $3SAT$ is modified by adding "dummy variables" and "dummy clauses", such that for the corresponding graph G the number of vertical lines is equal to the number of horizontal lines. K is set to the shortest length of a path connecting the two terminals of a net. Then in a set of paths corresponding to a satisfying truth assignment every path has length K. □

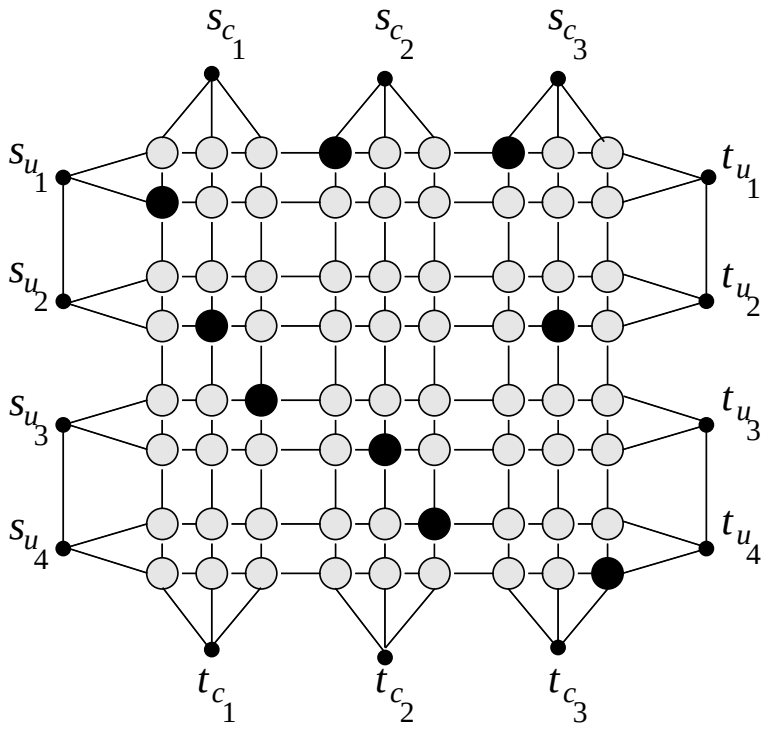


Figure 3: The instance of Problem 3.1 corresponding the clauses $c_1 = u_1 \vee u_2 \vee \neg u_3, c_2 = \neg u_1 \vee u_3 \vee \neg u_4, c_3 = \neg u_1 \vee u_2 \vee u_4$. The shaded super–vertices are of type 1, the black super–vertices of type 0.

4 Heuristics

In this section, we describe several heuristics for Problem 2.4 that are based on Algorithm 2.3. These heuristics have been implemented and included into the software package PlaNet [6]. We present an extensive experimental study comparing these heuristics and an exact method on more than 1800 instances. However, the exact method has exponential running time. For larger instances (more than 100 vertices and 20 nets) this method delivered no solution in more that 90% of the instances.

4.1 Description of the Heuristics

In principle, the heuristics pursue three different ideas. An obvious way to improve Algorithm 2.3 heuristically is to choose the start terminal x best possible. Secondly, the fact that only edges occupied by the auxiliary graph determined in Procedure 2.2 belong to the final solution can be used. Thirdly, shortest paths computations may be included into the determination of the edge–disjoint paths. Observe, that in general a collection of shortest paths connecting the terminals of the nets are not a feasible solution, i. e., are not pairwise edge-disjoint.

The first three heuristics use a sort of preprocessing for Algorithm 2.3 where the start terminal is chosen heuristically. Then the basic algorithm is called. The crucial fact used by the heuristics is that the $2k$–string of s–terminals and t–terminals on the outer face in counterclockwise ordering can be shifted cyclically without influencing the solvability of the instance. Possibly, s_i has to be exchanged with t_i to maintain the property that s_i occurs before t_i in the string. Now, an interval of length l_i is associated with every net n_i, and the list is shifted cyclically until the total interval length is minimal. After that, Algorithm 2.3 is called with the first terminal of the $2k$–string as the start terminal x. These heuristic algorithms also have linear running time.

Heuristic 4.1 Edge–Disjoint Min Interval Path Algorithm I
The interval length l_i is defined as the number of terminals between s_i and t_i in the sequence.

Heuristic 4.2 Edge–Disjoint Min Interval Path Algorithm II
The length l_i is defined as the number of edges on the outer face between s_i and t_i.

Heuristic 4.3 Edge–Disjoint Min Interval Path Algorithm III
The length l_i is defined as the number of edges on the outer face between s_i and t_i minus the edges incident to the terminals.

The next three heuristics also use a sort of preprocessing for Algorithm 2.3 where the start terminal is chosen heuristically. These heuristics start again at the ordering of the s– and t–terminals, in the first phase of the basic algorithm. But here the nets of the problem with parenthesis structure are considered. An interval length l_i is associated with every net $n_i^{()}$, $i \in \{1 \dots k\}$ of instance $(G, \mathcal{N}^{()})$. For the definition of l_i see below. Then, the $2k$-string of terminals with minimum total interval length is determined and Algorithm 2.3 is called with the first terminal as start terminal of that string. The running time is $\mathcal{O}(n + k^2)$.

Heuristic 4.4 Edge–Disjoint Min Parenthesis Interval Path Algorithm I
The interval length l_i is defined as the number of terminals between $s_i^{()}$ and $t_i^{()}$ in the sequence.

Heuristic 4.5 Edge–Disjoint Min Parenthesis Interval Path Algorithm II
The length l_i is defined as the number of edges on the outer face between $s_i^{()}$ and $t_i^{()}$.

Heuristic 4.6 Edge–Disjoint Min Parenthesis Interval Path Algorithm III
The length l_i is defined as the number of edges on the outer face between $s_i^{()}$ and $t_i^{()}$ minus the number of edges incident to the terminals.

The next two heuristics use the following observation. In the second phase of Algorithm 2.3 the paths are constructed using only edges from the auxiliary graph. The running time is linear respectively $\mathcal{O}(nk)$.

Heuristic 4.7 Reduced Edge–Disjoint Path Algorithm
The heuristic now uses the observation above by invoking Algorithm 2.3 first and then removing all edges, which are not considered. Then, a new start terminal is chosen randomly and the algorithm is called again with this reduced instance.

Heuristic 4.8 More Reduced Edge-Disjoint Path Algorithm
Heuristic 4.7 is applied for every terminal as the start terminal in the second call of the basic algorithm. This is done in the heuristic algorithm and the resulting solution is shown.

The following heuristics use a sort of postprocessing for the basic algorithm. Algorithm 2.3 is called first and then the constructed paths are reconsidered. The running time is linear respectively $\mathcal{O}(nk)$.

Heuristic 4.9 Edge–Disjoint Path Algorithm — Last Path Shortest Path
The aim of this heuristic algorithm is to shorten the length of the last path of the solution determined by Algorithm 2.3. Therefore, the last path is determined by an algorithm to compute shortest paths using only edges of the input instance which are not occupied by the other paths.

Heuristic 4.10 Edge–Disjoint Path Algorithm — Longest Path Shortest Path
Analogously, this heuristic algorithm shortens the longest path of the solution determined by Algorithm 2.3. The longest path is determined by an algorithm to compute shortest paths using only edges of the input instance which are not occupied by the other paths.

Heuristic 4.11 Edge–Disjoint Path Algorithm — All Paths Shortest Paths
This method uses a sort of postprocessing for the basic algorithm. Algorithm 2.3 is called first and then the constructed paths are reconsidered one after the other. Let $p_1, \ldots, p_k$ be the constructed paths. For p_k to p_1 all paths are removed from the graph and redetermined by an algorithm which determines shortest paths using only edges of the input instance which are not occupied by the other paths.

The last heuristic is a combination of two of the previously described methods. Its running time is $\mathcal{O}(nk)$.

Heuristic 4.12 Min Parenthesis Interval II and All Paths Shortest Paths
This method calls as a preprocessing Heuristic 4.5. As a postprocessing Heuristic 4.11 is executed.

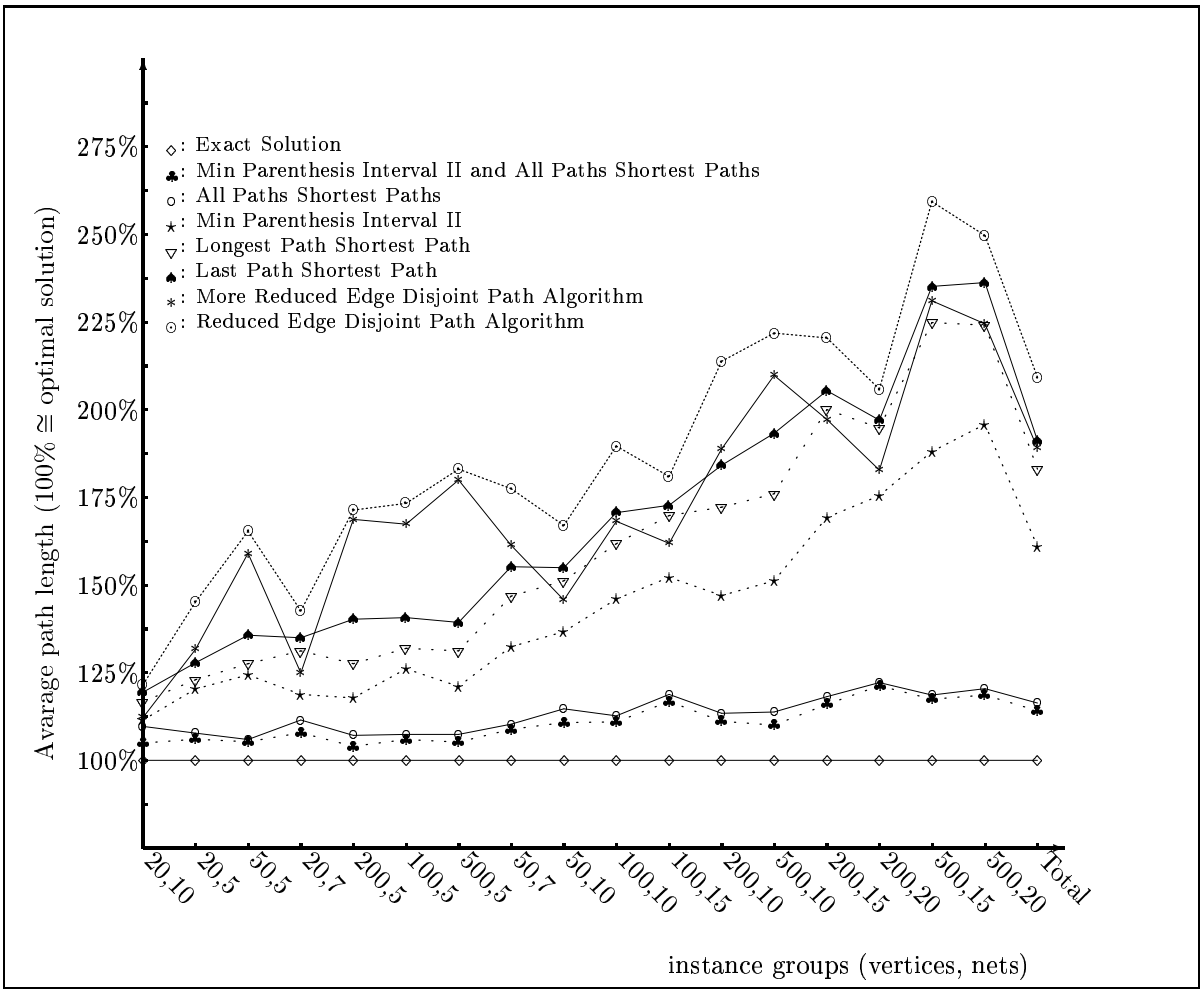


Figure 4: Representation of the avarage difference between the exact and heuritic solutions.

4.2 Experimental Studies

The heuristics presented in Section 4.1 have been implemented and included in the software package *PlaNet* [6]. PlaNet is a package to display and animate algorithms that work on planar networks. Its core is a collection of algorithms for various disjoint path problems. The aim of this package is to demonstrate these algorithms by means of a graphical user interface, as well as to support

the development of new algorithms. The package is implemented in C++ and runs on Sun Sparc stations, and the graphics are based on the X11 Window System. A reduced demo version and some additional information is available in the World Wide Web[2].

The heuristics were tested on randomly generated instances. The random instance generator applies the following two steps. First a triangulation is constructed. (Note that the triangulations are exactly the maximal planar graphs with respect to insertion of edges inside the outer face.) Afterwards, a couple of edges, paths, and cycles is removed. The Eulerian condition is guaranteed by removing a couple of edge–disjoint paths from the triangulation, namely such that the number of paths meeting a vertex is even if and only if this vertex already fulfills the evenness condition in the initial triangulation. Our experience is that suitable random distributions can be implemented this way much more easily than, for example, by constructing random graphs incrementally. In contrast, an incremental approach causes a lot of difficult technical problems (e.g., maintenance of planarity throughout the procedure).

We have studied the heuristics on more than 1800 instances in total. There are 100 instances of each size (n, k), where n is the number of vertices of the graph and k is the number of nets. The heuristics are compared to an exact approach [8] that is based on branch–and–bound. Of course, the exact method has exponential running time. In case of larger instances (more than 200 vertices and more than 20 nets) it did not contribute a solution for a large percentage of the instances. The results of our experimental studies are shown in Figure 4 and Table 1.

References

[1] M. Becker and K. Mehlhorn. Algorithms for routing in planar graphs. *Acta Informatica*, 23:163–176, 1986.

[2] M. Formann, D. Wagner, and F. Wagner. Routing through a dense channel with minimum total wire length. *J. of Algorithms*, 15:267–283, 1993.

[3] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. of Computer and System Sciences*, 30:209–221, 1985.

[4] M. Kaufmann and G. Klär. A faster algorithm for edge-disjoint paths in planar graphs. In W.-L. Hsu and R. Lee, editors, *ISA'91 Algorithms, Second International Symposium on Algorithms*, pages 336–348. Springer-Verlag, Lecture Notes in Computer Science, vol. 557, 1991.

[5] K. Matsumoto, T. Nishizeki, and N. Saito. An efficient algorithm for finding multicommodity flows in planar networks. *SIAM J. Comput.*, 14:289–302, 1985.

[6] G. Neyer, W. Schlickenrieder, D. Wagner, and K. Weihe. PlaNet — a demonstration package for algorithms on planar networks. Preprint 5, Universität Konstanz, 1996.

[7] H. Okamura and P. Seymour. Multicommodity flows in planar graphs. *J. of Combinatorial Theory, Series B*, 31:75–81, 1981.

[8] M. Oellrich. Master Thesis (german). TU Berlin 1996.

[9] D. Wagner. Optimal routing through dense channels. *Int. J. on Comp. Geom. and Appl.*, 3:269–289, 1993.

[10] D. Wagner and K. Weihe. A linear time algorithm for edge-disjoint paths in planar graphs. *Combinatorica*, 15:135–150, 1995.

[11] F. Wagner and B. Wolfers. Short wire routing in convex grids. In W.-L. Hsu and R. Lee, editors, *ISA'91 Algorithms, Second International Symposium on Algorithms*, pages 72–83. Springer-Verlag, Lecture Notes in Computer Science, vol. 557, 1991.

[2] URL `http://www.informatik.uni-konstanz.de/Research/Projects/PlaNet/`

vertices, nets	exact results	Min Parenthesis Interval II and All Paths Shortest	All Paths Shortest Paths	Min Parenthesis Interval II	Min Parenthesis Interval I	Min Interval II	Min Interval I	Min Parenthesis Interval III	Min Interval III	Longest Path Shortest Path	More Reduced	Last Path Shortest Path	Reduced
20,5	100.00	106.1	107.8	120.4	122.2	120.4	122.6	121.3	120.9	123.0	131.7	127.8	145.2
20,7	100.00	107.9	111.5	118.8	119.7	118.8	120.3	121.2	120.3	131.2	125.0	135.0	142.6
20,10	100.00	105.0	109.7	111.5	111.3	111.5	111.9	112.1	113.3	116.7	112.3	119.5	121.7
50,5	100.00	105.3	106.0	124.4	124.4	124.0	125.1	124.7	126.9	127.6	159.0	135.7	165.4
50,7	100.00	108.9	110.3	132.4	133.1	132.6	133.6	132.9	132.6	146.8	161.4	155.2	177.5
50,10	100.00	110.8	114.7	136.7	137.3	136.7	138.1	137.0	137.2	151.2	145.7	154.9	167.0
100,5	100.00	105.9	107.4	126.1	128.5	125.2	129.1	125.5	125.5	132.0	167.4	140.7	173.3
100,10	100.00	111.0	112.7	146.0	146.0	146.5	146.9	145.9	145.8	161.8	168.2	170.7	189.6
100,15	100.00	116.8	118.8	152.2	152.3	154.2	152.7	155.0	156.4	169.8	161.9	172.7	181.0
200,5	100.00	104.0	107.2	117.8	121.8	116.4	122.5	118.0	118.3	127.6	168.7	140.3	171.4
200,10	100.00	111.2	113.4	147.0	150.3	147.4	148.8	150.5	148.8	172.2	188.8	184.2	213.8
200,15	100.00	116.0	118.2	169.3	170.3	169.9	170.2	171.0	171.2	200.0	197.2	205.4	220.5
200,20	100.00⋆	121.3	122.2	175.5	175.5	175.5	176.1	177.1	177.1	194.8	182.8	197.0	205.8
500, 5	100.00	105.2	107.4	121.1	127.2	121.6	123.4	120.2	121.1	131.2	180.0	139.3	183.1
500,10	100.00	110.1	113.8	151.2	151.8	151.7	151.8	152.9	153.6	176.0	209.8	193.3	221.8
500,15	100.00	117.4	118.7	188.1	187.0	189.8	190.0	190.9	192.1	224.9	231.0	235.1	259.3
500,20	100.00◇	118.6	120.5	195.8	195.8	197.9	197.3	198.1	198.5	224.1	224.5	236.2	249.6
Total	100.00	114.2	116.3	161.0	161.6	161.8	162.4	162.7	163.1	183.1	189.0	191.2	209.2

Table 1: The average path length of the heuristics in percent in respect to the exact solutions.
⋆: Only 45% of all solvable instances were solved.
◇: Only 14% of all solvable instances were solved.