Acta Cybernetica **18** (2008) 503–527.

Two New Approximation Algorithms for the Maximum Planar Subgraph Problem*

Timo Poranen†

Abstract

The maximum planar subgraph problem (MPS) is defined as follows: given a graph G, find a largest planar subgraph of G. The problem is NP-hard and it has applications in graph drawing and resource location optimization. Călinescu et al. [J. Alg. 27, 269-302 (1998)] presented the first approximation algorithms for MPS with nontrivial performance ratios. Two algorithms were given, a simple algorithm which runs in linear time for bounded-degree graphs with a ratio 7/18 and a more complicated algorithm with a ratio 4/9. Both algorithms produce outerplanar subgraphs.

In this article we present two new versions of the simpler algorithm. The first new algorithm still runs in the same time, produces outerplanar subgraphs, has at least the same performance ratio as the original algorithm, but in practice it finds larger planar subgraphs than the original algorithm. The second new algorithm has similar properties to the first algorithm, but it produces only planar subgraphs. We conjecture that the performance ratios of our algorithms are at least 4/9 for MPS.

We experimentally compare the new algorithms against the original simple algorithm. We also apply the new algorithms for approximating the thickness and outerthickness of a graph. Experiments show that the new algorithms produce clearly better approximations than the original simple algorithm by Călinescu et al.

Keywords: maximum planar subgraph, maximum outerplanar subgraph, thickness, outerthickness, triangular cactus heuristic, approximation algorithm

1 Introduction

A graph is *planar* if it admits a plane drawing where no two distinct edges intersect apart from their endpoints, otherwise the graph is *non-planar*. Let $G = (V, E)$ be a

*Work funded by the Tampere Graduate School in Information Science and Engineering (TISE) and supported by the Academy of Finland (Project 51528). The results of this paper have originally published in the PhD thesis [32] of the author.

†Department of Computer Sciences, P.O. Box 607, FIN-33014 University of Tampere, Finland, E-mail: tp@cs.uta.fi

graph without loops and parallel edges. If a graph $G' = (V, E')$ is a planar subgraph of G such that every graph G'' obtained from G' by adding an edge from $E \setminus E'$ is non-planar, then G' is called a *maximal planar subgraph* of G. Let $G' = (V, E')$ be a maximal planar subgraph of G. If there is no planar subgraph $G'' = (V, E'')$ of G with $|E''| > |E'|$, then G' is a *maximum planar subgraph*. A maximal planar subgraph is maximal in the sense that adding edges is not possible and a maximum planar subgraph is maximal with respect to the cardinality of its edge set.

A planar graph is outerplanar if it admits a plane drawing where all its vertices lie on the same face and no two distinct edges intersect apart from their endpoints. *Maximal* and *maximum outerplanar subgraphs* are defined analogously.

Maximum planar subgraphs have applications in facility layout [17] and graph drawing [20, 22]. The problems of finding a maximum planar subgraph or a maximum outerplanar subgraph are denoted respectively throughout this work by MPS and MOPS. Both problems are known to be NP-hard [26, 38]. Therefore, heuristic algorithms are needed to find good approximations. Several methods for approximating MPS are given in the literature, see for example a survey by Liebers [25] and the references given there.

The *performance ratio* of an approximation algorithm for a maximization problem is the worst case ratio of solutions obtained to the cost of optimal solution. The performance ratio measures the solution quality of an approximation algorithm, the closer to 1 the ratio is, the better solutions the algorithm guarantees. A simple way to find an approximation for MPS is to produce a spanning tree for the input graph. Since a spanning tree of an n-vertex graph contains $n - 1$ edges and a maximum planar subgraph could contain at most $3n - 6$ edges, the performance ratio of this method is 1/3 [10].

Călinescu et al. [5] presented the first approximation algorithms with non-trivial performance ratios for MPS and MOPS. Their method, *triangular cactus heuristic*, gives a performance ratio of 4/9 for MPS and 2/3 for MOPS. These approximations can be achieved by a complicated algorithm having a running time of $O(m^{3/2} n \log^6 n)$ for a graph with n vertices and m edges. There is no known implementation of this algorithm. Călinescu et al. also presented a simple version of their algorithm having performance ratios 7/18 and 7/12 respectively. The simple algorithm runs in linear time for bounded-degree graphs.

In this paper we introduce two new algorithms based on the simple version of the algorithm presented by Călinescu et al. for MPS and MOPS. Our first algorithm also runs in linear time for bounded-degree graphs and it has at least the same performance ratio as the original simple algorithm. The second algorithm has properties similar to those of the first algorithm, but it produces only planar subgraphs. We conjecture that the new algorithms have at least the same performance ratio as the more complicated algorithm. Our experiments show that the new algorithms produce clearly better approximations than the original simple algorithm. Since the better algorithm by Călinescu et al. is difficult to implement, it is not included in our experiments.

The *thickness* of a graph is the minimum number of planar subgraphs into which the graph can be decomposed. The *outerthickness* of a graph is the minimum

number of outerplanar subgraphs into which the graph can be decomposed. The thickness and outerthickness are topological invariants that measure the graph's embeddability into the plane. Determining the thickness of a graph plays an important role in VLSI circuit design: the minimum number of planar subgraphs whose union is the graph corresponding to an electronic circuit provides an efficient way to find a decomposition for the distinct layers of the circuit [30].

Determining the thickness of a given graph is NP-hard [28] but the complexity status of determining the outerthickness is not known. The thickness is known for hypercubes [23], complete graphs [1, 2] and complete bipartite graphs [3]. Similar results for outerthickness have been reported by Guy and Nowakowski [14, 15].

Only one method to obtain approximations for thickness has been introduced in the literature: extract maximal planar subgraphs from the original graph until the remaining graph is planar [8, 30]. All earlier algorithms apply planarity tests to construct large planar subgraphs.

A new approach presented here for approximating the thickness of a graph is to extract planar subgraphs in such a way that the extracted graph is constructed without using planarity testing algorithms. In this paper we apply the simple algorithm by Călinescu et al. [5] with our new algorithms for approximating the thickness and outerthickness of a graph. Our experiments show that the new algorithms give better approximations than the original simple algorithm.

The rest of this paper is organised as follows. Next we give graph theoretical definitions and introduce a greedy algorithm for MPS with the extraction algorithm for the thickness problem. We also describe the triangular cactus heuristic. The new algorithms and their theoretical properties are discussed in Section 3. The experimental comparison of the algorithms for MPS is presented in Section 4 and then the algorithms are applied to the thickness problem in Section 5. The last section concludes our paper.

2 Preliminaries

2.1 Graph-theoretical definitions

For the basic graph-theoretical definitions, we refer to Harary [16]. Throughout this work we assume that graphs are simple and connected. An $m \times n$ *grid graph* is the product of paths of length m and n and contains mn vertices and $2mn - n - m$ edges.

A *triangular structure* is a graph in which every cycle is a triangle. A *triangular cactus* is a triangular structure in which every edge is in some cycle. A triangular structure is outerplanar, since the graph cannot contain a subdivision of K_4 or $K_{3,2}$.

A *maximal outerplanar graph* (mop) is an outerplanar graph such that inserting any edge produces a non-outerplanar graph. Next we present a useful characterization for mops [4] having at least three vertices.

Definition 2.1. *Mops having at least three vertices can be defined recursively as follows:*

1. *K_3 is a mop.*
2. *If G is a mop which is embedded in the plane so that every vertex lies on the outer face and G' is obtained by joining a new vertex to two vertices of an edge on the outer face of G, then G' is a mop.*
3. *H is a mop if and only if it can be obtained from K_3 by a finite sequence of applications of statement (2).*

2.2 A greedy algorithm for MPS

Throughout this work, all algorithms for MPS return a subgraph of the input graph. The cost of a solution is the number of edges in the returned approximation. Thickness algorithms return a partition of the edges of the input graph. The cost of a solution is the number of subsets in the partition.

A greedy algorithm to search for a maximal planar subgraph is to apply a planarity testing algorithm and to add as many edges as possible to a planar subgraph. See Algorithm 2.1 (GRE) for a detailed description of this edge adding method. The performance ratio of GRE is 1/3 for MPS [10].

GRE($G = (V, E), G' = (V, E')$)
1 $E'' = E \setminus E'$;
2 **while** there is an edge (u, v) in E''
3 **do** $E' \leftarrow E' \cup \{(u, v)\}$, $E'' \leftarrow E'' \setminus \{(u, v)\}$;
4 **if** (V, E') is not planar
5 **then** $E' \leftarrow E' \setminus \{(u, v)\}$;
6 **return** (V, E');

Algorithm 2.1: GRE for MPS.

GRE takes as input a graph $G = (V, E)$ and its planar subgraph $G' = (V, E')$. The algorithm returns a maximal planar subgraph containing the input graph as a subgraph. Our reason for assuming that a planar subgraph is given as input to the algorithm is that then we can apply GRE to improve solutions of other heuristics. This approach is described in Section 3. The running time of GRE heuristic is $O(|V||E|)$ if a linear time planarity testing algorithm [18] is applied at Step 4.

2.3 The thickness heuristic

Next we describe the basic approach to obtain approximations for thickness. The extraction method was first studied by Cimikowski [8] and Mutzel et al. [30]. For

a detailed description of the extracting method see Algorithm 2.2 (THICK). Step 3 of the algorithm is usually given as "find a maximal/maximum planar subgraph" instead of finding just a planar subgraph.

THICK($G = (V, E)$)
1 $P \leftarrow \emptyset$; $t \leftarrow 1$;
2 **while** $E \neq \emptyset$
3 **do** find a planar subgraph $G' = (V, E_t)$ of G;
4 $E \leftarrow E \setminus E_t$;
5 $P \leftarrow P \cup \{E_t\}$;
6 $t \leftarrow t + 1$;
7 **return** P;

Algorithm 2.2: Basic structure of the extraction algorithm for the thickness problem.

THICK takes as input a graph $G = (V, E)$ and it returns a partition of the edges into subsets inducing planar subgraphs. The running time of Algorithm 2.2 depends heavily on the method used in Step 3. If a maximal planar subgraph is recognised from the input graph by applying GRE, the running time of the algorithm is $O(|V|^2|E|)$.

2.4 Triangular cactus heuristics

Next we introduce the triangular cactus algorithms for MPS and MOPS [5]. Given a connected graph $G = (V, E)$, the triangular cactus heuristic is based on finding a subgraph $G' = (V, E')$ whose components are triangular cacti. The subgraph is constructed in the following way: E' is initialized to be empty. Triangles having all vertices in different components in G' are searched from G and added to E'. After all suitable triangles have been added to G', the subgraph is connected by adding edges until the resulting graph is a connected triangular structure. See Algorithm 2.3 (CA) for a detailed description of the triangular cactus heuristic. Steps 2 and 3 are called Phase 1 (the construction phase of a triangular cactus) and Steps 4 and 5 are called Phase 2 (the connection phase) of the algorithm.

Algorithm CA can be implemented to run in linear time as shown by Călinescu et al. [5], provided that the maximum degree of the graph is bounded by a constant. The theorem below concludes the properties of CA.

Theorem 2.2. *[5]* CA *runs in linear time for bounded-degree graphs. The performance ratio of* CA *for MPS is* 7/18 *and for MOPS* 7/12.

If a maximum triangular cactus is searched for in Phase 1 instead of the triangular cactus of CA, the performance ratio increases to 4/9 for MPS and 2/3 for MOPS [5]. The algorithm is denoted by CA_M.

```
CA(G = (V, E))
1  E' ← ∅;
2  while there is a triangle (v1, v2, v3) in G such that
   v1, v2 and v3 belong to different components of (V, E')
3        do E' ← E' ∪ {(v1, v2), (v2, v3), (v3, v1)};
4  while there is an edge (v1, v2) ∈ E such that v1 and v2 belong to
   different components in (V, E')
5        do E' ← E' ∪ {(v1, v2)};
6  return (V, E');
```

Algorithm 2.3: CA for MPS and MOPS.

All the known algorithms for finding a maximum triangular structure are very complicated. The method proposed by Călinescu et al. [5] was based on reducing the problem of finding a maximum triangular structure to a graphic matroid parity problem [27] and then solving it with an algorithm by Gabow and Stallman [11]. This method leads to running time $O(m^{3/2}\log^6 n)$. There are no known implementations of the algorithm. The following theorem formulates the properties of CA_M.

Theorem 2.3. *[5] The performance ratio of* CA_M *for MPS is* $4/9$ *and for MOPS* $2/3$. CA_M *runs in* $O(m^{3/2}\log^6 n)$ *for a graph with* m *edges and* n *vertices.*

3 New algorithms for MPS and MOPS

In this section we introduce first our new algorithms, CA1 for MPS and MOPS and CA2 for MPS. We also study the theoretical properties of the algorithms and compare them with CA and CA_M.

When a triangle is found in CA, it always connects three vertices from different components of the subgraph. It is easy to see that not all the vertices of a triangle need belong to different components. It is enough to have two vertices v_1 and v_2 joined by an edge (v_1, v_2) in one component and the third vertex v_3 in another component forming a triangle (v_1, v_2, v_3). When triangles are added using this principle whenever possible, and otherwise requiring that the vertices of the triangle belong to different components, the planarity is not violated. If any triangle is added with this new principle, the resulting graph is no longer a triangular structure. To ensure that the constructed subgraph is also outerplanar, it is necessary and sufficient to demand that (v_1, v_2) belongs to at most two triangles at the same time. The algorithm applying this restriction and producing outerplanar subgraphs is denoted by CA1, and the algorithm without the restriction is denoted by CA2. The properties of CA1 are studied first. The exact description of CA1 is given in Algorithm 3.1.

CA1(G=(V,E))

1 $E' \leftarrow \emptyset$;
2 **repeat** **while** there is a triangle (v_1, v_2, v_3) in G such that (v_1, v_2) belongs to exactly one triangle in E' and v_3 to a different component of (V, E')
3 **do** $E' \leftarrow E' \cup \{(v_2, v_3), (v_3, v_1)\}$;
4 **if** there is a triangle (v_1, v_2, v_3) in G such that v_1, v_2 and v_3 belong to different components of (V, E')
5 **then** $E' \leftarrow E' \cup \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$;
6 **until** the number of edges in E' does not increase during the loop;
7 **while** there is an edge $(v_1, v_2) \in E$ such that v_1 and v_2 belong to different components in (V, E')
8 **do** $E' \leftarrow E' \cup \{(v_1, v_2)\}$;
9 **return** (V, E');

Algorithm 3.1: CA1 for MPS and MOPS.

CA1 was inspired by the recognition algorithm for maximal outerplanar graphs proposed by Mitchell [29]. The algorithm was based on extracting degree 2 vertices from the graph. In CA1, vertices of degree 2 are added to an outerplanar subgraph.

Figure 1 provides an illustration of the behaviour of CA and CA1 for the graph cimi-g4 [9], which is a non-planar graph with 10 vertices and 22 edges. A maximum planar subgraph of this graph contains 20 edges. The triangles are numbered in the order they are found. This order depends on the implementation of the algorithm and the representation of the graph. CA first finds four triangles and then it connects one remaining vertex with the rest of the subgraph. The planar subgraph contains 13 edges. CA1 finds first one triangle, then it adds 5 triangles that increase the number of edges by 2 and finally a triangle with three new edges is added. The size of the planar subgraph is now 16.

Next we show that CA1 can be implemented to run in linear time, if the maximum degree of the input graph is bounded by a constant.

Lemma 3.1. CA1 *runs in linear time for bounded-degree graphs.*

Proof. To establish that CA1 runs in linear time, it is sufficient to note that the steps where a triangle connecting two vertices from the same component and one vertex from another component take in total linear time provided that the degree of the graph is bounded. The total time for all other operations is linear for bounded-degree graphs by Theorem 2.2.

Suppose that the degree of a graph is bounded by a constant d. Each time an edge (v_1, v_2) is considered in Step 2, it takes at most d^2 time to check the adjacency lists of v_1 and v_2 to recognise a triangle. Since it is enough to consider each edge only once in the first while loop, CA1 runs in time $O(n)$ for bounded-degree graphs.

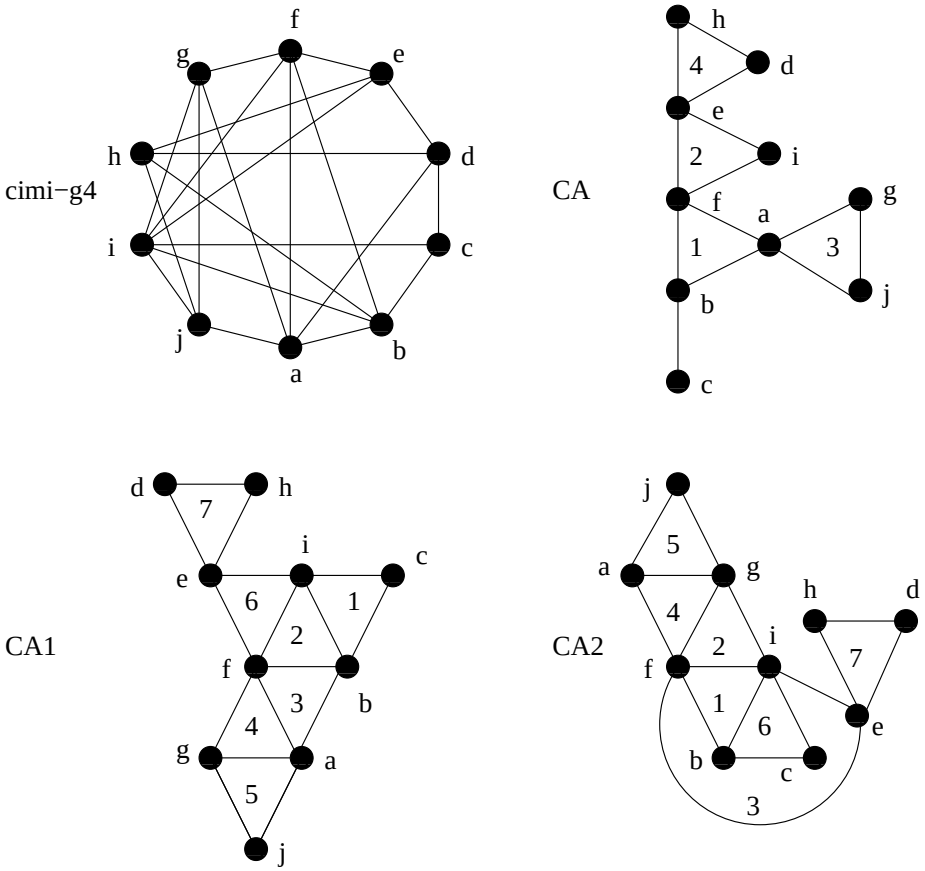

Figure 1: Illustrations of planar subgraphs for graph cimi-g4 found by CA, CA1 and CA2. Triangles are enumerated in the order they have been found in a sample run.

□

To show that the performance ratio of CA1 for MPS is at least 7/18 and for MOPS at least 7/12, the original proof of Theorem 2.2 can be applied directly. We only outline the important property of CA1 that makes it possible to apply the proof technique introduced by Călinescu et al. [5].

Lemma 3.2. *The performance ratio of* CA1 *for MPS is at least* 7/18 *and for MOPS at least* 7/12*.*

Proof. Let G_{CA} and G_{CA1} be the planar subgraphs produced by CA and CA1 after Phase 1 respectively. No triangle was added to G_{CA} if two of its vertices were in the same component. The same holds for G_{CA1}: there is no triangle in the input

graph with its vertices in different components in G_{CA1}. The original proof was based on this observation, and therefore, it follows that CA1 has at least the same performance ratio as CA. □

The proof of the upper bound given by Călinescu et al. [5] for the performance ratio of CA cannot be applied to CA1, but it is clear that the ratio cannot exceed 1/2, as shown by the following constructive proof.

Lemma 3.3. *The performance ratio of* CA1 *for MPS is at most* 1/2.

Proof. Let G be an $n \times n$ grid graph with $n \geq 2$. The graph has in total n^2 vertices and $2n^2 - 2n$ edges. Since G is planar, the maximum planar subgraph is the graph itself. CA1 finds a planar subgraph with $n^2 - 1$ edges by constructing a spanning tree of G. The ratio between the number of edges found by CA1 and the number of edges in G is

$$\frac{n^2 - 1}{2n^2 - 2n}.$$

The limit of the ratio is 1/2 as n tends to infinity.

□

Next we present a sample graph which shows that the performance ratio of CA1 for MOPS is at most 2/3.

Lemma 3.4. *The performance ratio of* CA1 *for MOPS is at most* 2/3.

Proof. Let G be a $2 \times n$ grid graph. G has in total $2n$ vertices and $3n - 2$ edges. Since G is outerplanar, the maximum outerplanar subgraph is the graph itself. CA1 finds an outerplanar subgraph with $2n - 1$ edges by constructing a spanning tree of G. The ratio between the number of edges found by CA1 and the number of edges in G is

$$\frac{2n - 1}{3n - 2}.$$

The limit of the ratio is 2/3 as n tends to infinity. □

We can now conclude the properties of CA1 for MPS and MOPS.

Theorem 3.5. *The performance ratio of* CA1 *for MPS is at least* 7/18 *and at most* 1/2. *The performance ratio of* CA1 *for MOPS is at least* 7/12 *and at most* 2/3. *The algorithm runs in linear time for bounded-degree graphs.*

There is a gap between the lower and upper bounds of the performance ratios of CA1 for MPS and MOPS, and the exact performance ratio is left open. One way to confirm or refute that the performance ratio is at least 4/9 for MPS is to show that a subgraph produced by CA1 has always at least the same number of edges as a maximum triangular structure of a graph. We present conjecture for the performance ratio of CA1 for MPS and MOPS. The computational experiments reported in the next section support the conjecture.

Conjecture 3.6. *The performance ratio of* CA1 *for MPS is at least* 4/9 *and for MOPS exactly* 2/3*.*

Next we study CA2. From the condition in the first while loop of CA1 it follows that at the end of the algorithm an edge of G' belongs at most to two triangles. It is not necessary in the case of planar subgraphs to require that one edge should belong to at most two triangles at the same time. The restriction "(v_1, v_2) belongs to exactly one triangle in E'" of the while loop of CA1 can be changed to "(v_1, v_2) belongs to E'". This observation leads to Algorithm CA2. Now outerplanarity is violated if at the end of the algorithm any edge belongs to more than two triangles (a forbidden subgraph $K_{3,2}$ is created [16]). The subgraph remains planar. In Figure 1, there is an illustration of the behaviour of CA2. Note that the edge (f, i) belongs to three triangles and hence, outerplanarity is violated. The planar subgraph found by CA2 contains 16 edges.

CA2($G = (V, E)$)
1 $E' \leftarrow \emptyset$;
2 **repeat** **while** there is a triangle (v_1, v_2, v_3) in G such that (v_1, v_2) belongs to E' and v_3 to a different component of (V, E')
3 **do** $E' \leftarrow E' \cup \{(v_2, v_3), (v_3, v_1)\}$;
4 **if** there is a triangle (v_1, v_2, v_3) in G such that v_1, v_2 and v_3 belong to different components in (V, E')
5 **then** $E' \leftarrow E' \cup \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$;
6 **until** the number of edges in E' does not increase during the loop;
7 **while** there is an edge $(v_1, v_2) \in E$ such that v_1 and v_2 belong to different components in (V, E')
8 **do** $E' \leftarrow E' \cup \{(v_1, v_2)\}$;
9 **return** (V, E');

Algorithm 3.2: CA2 for MPS.

The linear running time of CA2 for bounded-degree graphs follows directly by Theorem 2.2 for CA and by Lemma 3.1 for CA1. The bounds for the performance ratio of CA2 are the same as they are for CA1. The following theorem concludes the properties of CA2.

Theorem 3.7. *The performance ratio of* CA2 *for MPS is at least* 7/18 *and at most* 1/2*, and the algorithm runs in linear time for bounded-degree graphs.*

We give a similar conjecture for the performance ratio of CA2 as for CA1. This conjecture is also supported by the experiments reported in the next section.

Conjecture 3.8. *The performance ratio of* CA2 *for MPS is at least* 4/9*.*

Next we present three simple corollaries that describe the properties of the algorithms.

A difference between CA1, CA2, CA and CA_M is that CA1 and CA2 recognise maximal outerplanar graphs. This follows directly from Definition 2.1, which gave a recursive method to construct a maximal outerplanar graph.

Corollary 3.9. CA1 *and* CA2 *recognise maximal outerplanar graphs.*

The second corollary yields a graph class for which CA1 and CA2 find better approximations than CA_M.

Corollary 3.10. *There are graphs for which the limit of the ratio of the solutions of* CA1 *(or* CA2*) and* CA_M *is* 4/3*.*

Proof. Let G be a maximal outerplanar graph with n vertices. G has $2n-3$ edges. Since CA1 (CA2) finds all edges of a maximal outerplanar graph and a maximum triangular structure of a maximal outerplanar graph contains $3\lfloor(n-1)/2\rfloor$ edges [5], the ratio of the solutions of CA1 (CA2) and CA_M is 4/3 as n tends to infinity. □

Our third corollary describes the differences between CA2 and CA1 (CA_M).

Corollary 3.11. *There are graphs for which the limit of the ratio of the solutions of* CA2 *and* CA1 *(or* CA_M*) is* 2*.*

Proof. Let G be a graph with a single triangle containing vertices v_1, v_2 and v_3. Add to G a new vertex v_i, where $i > 3$, and two edges (v_i, v_1) and (v_i, v_2). Continue this process and denote the graph by G'. If G' has k vertices, it has $2(k-2)+1$ edges. Since CA2 finds all edges of G' and CA1 (CA_M) finds $k+1$ (k) edges, the ratio of the solutions of CA2 and CA1 (CA_M) is 2 as n tends to infinity. □

CA, CA1 and CA2 can be made greedy by giving the subgraph constructed in Phase 1 as input to GRE. These greedy versions are denoted by GCA, GCA1 and GCA2. Since GRE connects the subgraph, at least the same number of edges is added as in Phase 2 of CA, CA1 and CA2. Therefore, GCA, GCA1 and GCA2 have the same performance ratios as CA, CA1 and CA2 respectively.

4 MPS experiments

In this section, different algorithms for MPS are compared. More detailed comparison statistics for the algorithms can be found in [32].

4.1 MPS algorithms and comparison measures

We implemented the following algorithms for MPS: CA, CA1, CA2 and their greedy versions GCA, GCA1 and GCA2 with the pure greedy algorithm GRE. The results of CA and CA1 are valid for MOPS.

Algorithms CA, CA1 and CA2 were randomized by always choosing the edges and start vertices randomly. The greedy heuristics were randomized by handling the edges in a random order.

Table 1: Test graph statistics for MPS.

Graph data				The best solutions						
graph	$\|V\|$	$\|E\|$	ub	CA	CA1	CA2	GRE	GCA	GCA1	GCA2
cimi-g1	10	21	19*	11	13	13	19	19	19	19
cimi-g2	60	166	165*	88	117	117	165	165	165	165
cimi-g3	28	75	73*	38	49	49	73	73	73	73
cimi-g4	10	22	20*	13	16	16	20	20	20	20
cimi-g5	45	85	82*	59	73	73	82	82	82	82
cimi-g6	43	63	59*	42	42	42	59	59	59	59
g10	25	71	69*	36	47	47	69	69	69	69
g11	25	72	69*	36	47	47	69	69	69	69
g12	25	90	69*	36	47	47	67	66	67	67
g13	50	367	144	73	97	97	119	120	128	125
g14	50	491	144	73	97	97	127	132	134	133
g15	50	582	144*	73	97	97	133	136	138	137
g16	100	451	294	137	162	167	175	193	200	196
g17	100	742	294	147	194	196	196	224	237	229
g18	100	922	294	147	197	197	210	230	244	239
g19	150	1064	444	218	274	283	266	305	326	323
rg100.1	100	261	260	119	124	125	150	157	157	157
rg100.2	100	271	270	118	125	127	151	160	162	162
rg100.3	100	297	294	120	128	128	153	163	163	164
rg100.4	100	334	294	126	136	140	155	172	175	174
rg100.5	100	373	294	137	153	153	162	186	186	186
rg150.1	150	387	386	171	174	175	214	222	223	223
rg150.2	150	402	401	176	182	182	213	224	225	226
rg150.3	150	453	444	171	179	179	221	237	230	232
rg150.4	150	473	444	180	190	190	217	236	241	238
rg150.5	150	481	444	178	185	185	221	237	236	236
rg200.1	200	514	513	222	227	227	270	278	283	280
rg200.2	200	519	518	216	219	219	268	274	277	277
rg200.3	200	644	594	235	243	244	280	303	306	309
rg200.4	200	684	594	237	254	261	282	308	317	317
rg200.5	200	701	594	235	251	253	285	311	314	314
rg300.1	300	814	813	324	330	330	390	402	406	407
rg300.2	300	1159	894	355	376	377	412	455	461	461
rg300.3	300	1176	894	360	376	378	411	457	461	464
rg300.4	300	1474	894	389	422	426	432	497	508	509
rg300.5	300	1507	894	400	428	430	438	504	515	512
tg100.1	100	300	294*	138	188	197	292	292	294	290
tg100.3	100	324	294*	142	191	197	264	290	284	283
tg100.5	100	344	294*	138	187	197	251	262	268	272
tg100.7	100	364	294*	138	191	197	236	255	262	276
tg100.9	100	384	294*	140	189	196	226	260	263	272
tg200.1	200	604	594*	275	375	397	582	594	592	594
tg200.3	200	624	594*	279	382	397	558	579	592	586
tg200.5	200	644	594*	275	373	397	515	551	569	578
tg200.7	200	664	594*	275	372	397	492	552	578	589
tg200.9	200	684	594*	279	377	397	487	543	558	566

* Upper bound is known to be optimal.

All algorithms were written in C++ and their source codes are available as part of the the program *apptopinv* [31]. LEDA 4.3 [24] was used for the planarity test. All test runs were executed on a computer (1992 BogoMips) which has one AMD Athlon $1GHz$ processor with 256 Megabytes memory running under Linux Mandrake 8.1.

CA, CA1, CA2, GRE, GCA, GCA1 and GCA2 were repeated 100 times for graphs with no more than 100 edges and 25 times for larger graphs.

To compare the algorithms, we concentrated on studying the running time and performance differences between the algorithms. Methods and measures for the experimental analysis of the heuristics used in this work are mainly given by Golden and Stewart [12].

Running times for the algorithms were obtained by running all test runs as background processes and performing the *time* command to obtain the total running time. Finally, this total running time was divided by the number of repeats to obtain the average running time of a run.

For each algorithm, it is easy to select the best solution from all repeats for a test instance. We can then count the total number of best solutions for each algorithm, that is, an algorithm is awarded 1 point, if it obtained the best or tied the best solution for a test instance among all the algorithms.

Another measure is the total number of points for an algorithm: a heuristic is awarded p points, if it obtained the pth best solution for an instance. The average rank of an algorithm is the total number of points divided by the number of test instances.

4.2 Test graph set for MPS

Since MPS is a much studied optimization problem, there is already a wide variety of suitable test graphs. We mainly used the same test graph set as Resende and Ribeiro [35].[1]

The test graph set used in this work contains 46 graphs. Statistics for the graphs are given in Table 1. For all graphs we have listed the name of the graph (graph) and the number of vertices ($|V|$) and edges ($|E|$). Then we give the upper bound for MPS (ub). If the upper bound is known to be optimal, it is marked with a star ($\star$). Finally, the best solution found over all runs for the heuristics is given.

For graphs with an unknown optima, the upper bound was obtained by applying Euler's polyhedron formula [16]. If the number of edges was less than the bound obtained from the formula, the upper bound is the number of edges decreased by one for non-planar graphs.

The first six graphs (cimi-g1 – cimi-g6) in Table 1 were taken from the experiments of Cimikowski [7]. These graphs have relevance to applications or have their origin in other research papers. Graphs cimi-g4, cimi-g5 and cimi-g6 were introduced originally in [19], [21] and [37] respectively. Graph cimi-g6 does not contain

[1] The graphs can be downloaded from `http://www.research.att.com/~mgcr/data/planar-data.tar.gz` (April 27, 2006).

any triangles. Graphs g10 – g19 are Hamiltonian graphs constructed by Goldsmith and Takvorian [13].

Graphs rg100.1 – rg300.5 are random graphs with the number of vertices varying between 100 and 300 and the number of edges varying between 261 and 1507. Table 1 also contains graphs with a planar subgraph of maximum size (tg100.1 – tg200.9). The graphs were generated by Cimikowski [7].

4.3 Comparison of CA, CA1 and CA2 for MPS

The best solutions for the heuristics are reported in Table 1. The difference in the performance of the fast algorithms is clear. CA1 and CA2 find quite similar solutions, and they outperform CA with a clear margin. For all 46 test instances, algorithm CA2 finds the best solution, and CA1 finds the same solution as CA2 for 22 graphs. The solutions of CA are inferior to those of CA1 and CA2 for graphs that contain triangles. The only graph for which all the algorithms found the same solution was cimi-g6. The average rank of the heuristics is 2.96 for CA, 1.52 for CA1 and 1.00 for CA2. The comparison statistics are collected in Table 2.

Table 2: Comparison of the performance of the fast MPS heuristics.

	CA	CA1	CA2
Number of times heuristic is the best or tied for the best	1	22	46
Average rank	2.96	1.52	1.00

Figure 2 shows the average running times of one run for CA, CA1 and CA2 as a function of the number of edges. For graphs having more than 1600 edges, the running times are taken from the graphs used in the thickness algorithms comparison given in Section 5. Further, Figure 2 has the average running times of the greedy heuristics to illustrate the running time differences.

The running times of CA, CA1 and CA2 are less than one tenth of a second for all graphs up to 1600 edges. For the largest graphs, $r9$ with 449550 edges, used in the thickness comparison, we obtained the following average running times for the heuristics: 4.2, 5.8 and 6.5 seconds for CA, CA1 and CA2 respectively.

The running time differences between CA, CA1 and CA2 are in general very small. Only with graphs having more than 10000 edges can it be seen that CA is slightly faster than the other two algorithms. The sharp turns in the curve are the influence of the different ratios of the number of vertices and edges in a test graph. All three heuristics run faster for a sparse graph than for a dense graph with the same number of edges.

To further compare CA2 and CA we studied the relative differences of their solutions. See Figure 3 for the ratios of the poorest solutions (see [32] for these results) of CA2 and the best solutions of CA. The worst solutions by CA1 and CA2

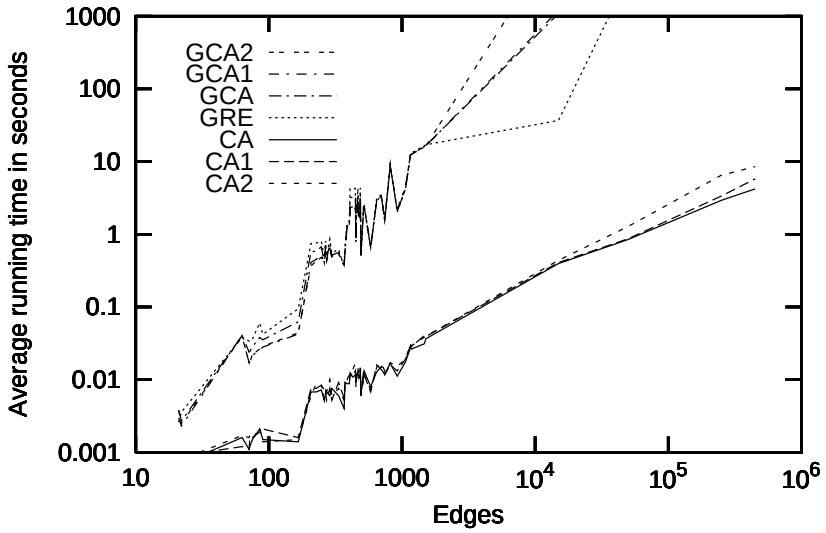

Figure 2: Average running times of MPS heuristics. Notice that the axes are logarithmic.

were always at least as good as the best solution by CA. The greatest improvement was for tg200.1 with a 1.44 times better solution when CA2 was used instead of CA. In general, the greatest improvements were obtained for graphs containing a planar subgraph of maximum size (tg100.1 – tg200.9). The solutions of CA2 were on average 20 percentages better than those of CA.

The worst case ratios of CA solutions and the optimal (marked with a dot) or the best known (marked with a circle) solution [32] are shown in Figure 4. Our experiments give evidence on the conjectured performance ratio 4/9 for the new algorithms: the solutions by CA1 and CA2 were never more than 4/9 away from the optima. For the ratios of the poorest found solutions and the optimal or the best known solution for CA2, see Figure 5. The solutions of CA1 and CA2 were never less than 0.61 and 0.65 times the optima respectively.

4.4 Comparison of GRE, GCA, GCA1 and GCA2 for MPS

It is clear that when a greedy method to add edges is applied instead of just connecting the subgraphs in Phase 2 of CA, CA1 and CA2, the solutions remain at least the same. The main questions are, thus, how much the greedy approach improves the solutions, how much longer running time is needed, and if there are graphs for which CA, CA1 or CA2 outperform GRE.

As shown in Subsection 4.3, CA2 outperformed CA1, but when the greedy algorithms were considered, GCA1 produced approximations similar to GCA2.

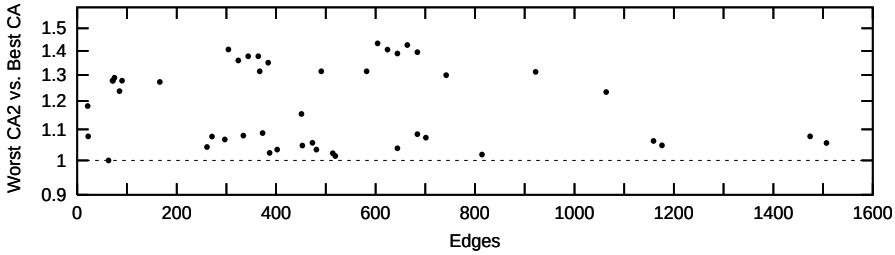

Figure 3: Ratios of the worst solutions of CA2 and the best solutions of CA.

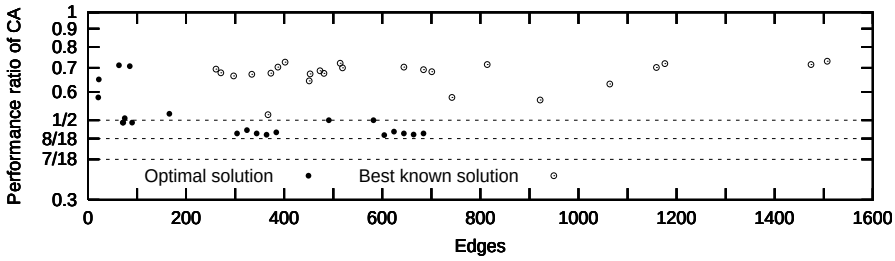

Figure 4: Ratios of the worst solutions of CA and the optimal or the best known solution.

GCA1 found the best or the tied best solution for 30 and GCA2 for 31 from 46 test instances, but the average ranks for these two heuristics were both 1.37. One explanation for the success of GCA1 is that the method of constructing a solution in Phase 1 of CA2 is greedier than that in CA1. The solution of CA2 could contain more edges than that of CA1, but it is more difficult to insert additional edges into the subgraph. GCA and GRE found the best or tied best solutions for 13 and 9 graphs and the average ranks were 2.41 and 3.41 respectively. These results are listed in Table 3.

Table 3: Comparison of the performance of the greedy MPS heuristics.

	GRE	GCA	GCA1	GCA2
Number of times heuristic is the best or tied for the best	9	13	30	31
Average rank	3.41	2.41	1.37	1.37

The running time differences of the greedy heuristics are in general very small as shown in Figure 2. For graphs with fewer than 1600 edges, GCA1 and GCA2 are

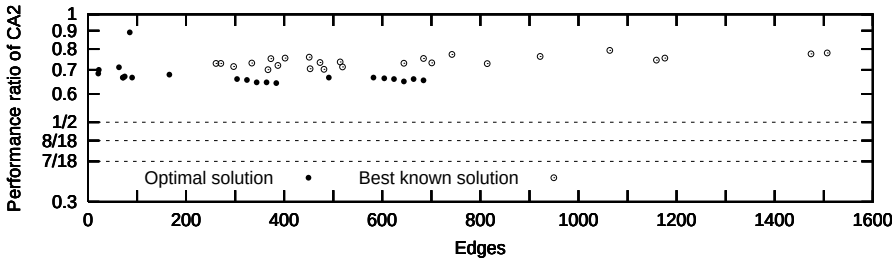

Figure 5: Ratios of the worst solutions of CA2 and the optimal or the best known solution.

the fastest heuristics and their average running time curves coincide. For the larger graphs (running times are taken from graphs used in the thickness comparison), GRE seems to be the fastest by a small margin.

We recognised test instances for which the new algorithms outperformed GRE. CA1 and CA2 found better solutions for graph g19. This shows that CA, CA1 and CA2 can find better solutions for large and sparse graphs than GRE in significantly shorter computation time. This coincides with the theoretical properties of CA1, CA2 and GRE.

CA did not find better solutions than GRE in our tests, but it has been reported that CA sometimes achieves better approximations for graphs with density varying between 0.03 and 0.15 when the algorithms are applied for MOPS [33].

GCA2 improves the solutions of CA2 on average by 30 percent. The same holds for GCA and GCA1.

5 Thickness experiments

5.1 Thickness algorithms and their implementation

For the thickness problem, we tested the extraction algorithm THICK by applying in Step 4 algorithms CA, CA1, CA2, GCA, GCA1, GCA2 and GRE. Also, we implemented an ST heuristic which in each iteration extracts the set of tree-edges found by a depth-first search. In what follows, these algorithms are simply denoted by the name of the extraction method. All these algorithms approximate thickness, but algorithms ST, CA and CA1 directly produce approximations for outerthickness.

ST, CA, CA1 and CA2 were repeated 25 times for graphs with fewer than 2000 edges, 10 times for graphs having more than 2000 edges but not more than 250000 edges and 5 times for larger graphs. Greedy heuristics for the thickness were applied only for graphs with 79800 edges or less. The number of repetitions was 25 for graphs with fewer than 2000 edges and 10 times for graphs with fewer

Table 4: Test graph statistics for the thickness problem.

Graph data				The best solutions							
graph	$\|V\|$	$\|E\|$	lb	ST	CA	CA1	CA2	GRE	GCA	GCA1	GCA2
K_{10}	10	45	$3^\star$	5	4	3	4	3	3	3	3
K_{15}	15	105	$3^\star$	8	7	5	5	4	4	4	4
K_{20}	20	190	$4^\star$	11	9	7	6	5	5	5	5
K_{30}	30	435	$6^\star$	16	13	10	9	8	7	7	7
K_{40}	40	780	$7^\star$	21	18	12	12	11	9	9	9
K_{50}	50	1225	$9^\star$	27	22	15	15	13	12	11	11
K_{60}	60	1770	$11^\star$	32	27	18	18	16	14	13	13
K_{70}	70	2415	$12^\star$	38	32	21	21	19	16	15	15
K_{80}	80	3160	$14^\star$	43	36	24	24	21	19	17	17
K_{90}	90	4005	$16^\star$	48	41	27	27	24	21	19	20
K_{100}	100	4950	$17^\star$	54	45	30	31	27	23	21	22
K_{150}	150	11175	$26^\star$	81	69	45	45	42	39	35	34
K_{200}	200	19900	$34^\star$	108	91	60	60	56	52	47	47
K_{300}	300	44850	$51^\star$	164	137	92	90	84	79	71	72
K_{400}	400	79800	$67^\star$	217	183	121	120	112	105	96	98
K_{500}	500	124750	$84^\star$	271	230	152	149	-	-	-	-
K_{600}	600	179700	$101^\star$	334	277	185	186	-	-	-	-
K_{700}	700	244650	$117^\star$	379	320	210	218	-	-	-	-
K_{800}	800	319600	$134^\star$	437	366	247	250	-	-	-	-
K_{900}	900	404550	$151^\star$	490	412	276	281	-	-	-	-
K_{1000}	1000	499500	$167^\star$	551	456	303	315	-	-	-	-
$r_{20,92}$	20	92	$2^\star$	6	4	4	4	2	3	3	3
$r_{40,311}$	40	311	3	9	7	6	5	4	5	4	4
$r_{60,556}$	60	556	4	10	8	7	7	6	5	5	5
$r_{80,939}$	80	939	5	13	10	8	8	7	7	6	6
$r_{100,1508}$	100	1508	5	17	13	10	10	9	8	8	8
r0	1000	14985	6	17	15	14	14	14	12	12	12
r1	1000	49950	17	53	43	32	31	36	30	27	27
r2	1000	99900	34	103	89	57	57	-	-	-	-
r3	1000	149850	51	160	130	82	83	-	-	-	-
r4	1000	199800	67	213	177	108	110	-	-	-	-
r5	1000	249750	84	264	224	133	138	-	-	-	-
r6	1000	299700	101	312	270	158	170	-	-	-	-
r7	1000	349650	117	363	316	184	198	-	-	-	-
r8	1000	399600	134	413	361	209	230	-	-	-	-
r9	1000	449550	151	465	411	235	261	-	-	-	-
rr1	1000	5000	2	6	6	6	6	5	5	5	5
rr2	1000	25000	9	26	22	19	19	20	17	16	16
rr3	1000	50000	17	51	43	32	31	36	30	27	27
rr4	1000	75000	26	76	65	44	44	50	43	37	37
rr5	1000	100000	34	101	88	57	57	-	-	-	-
rr6	1000	125000	42	126	112	69	70	-	-	-	-
rr7	1000	150000	51	151	136	82	83	-	-	-	-
rr8	1000	175000	59	176	160	94	96	-	-	-	-
rr9	1000	200000	67	201	184	107	108	-	-	-	-
rr10	1000	225000	76	226	209	119	121	-	-	-	-
rr11	1000	250000	84	251	233	132	134	-	-	-	-

* Lower bound is known to be optimal.

$^-$ The algorithm is not applied for this graph.

than 5000 edges. For larger graphs only one run was performed. The comparison measures given in Section 4 also hold for the thickness experiments.

5.2 Test graph set for thickness

Algorithms for the thickness problem are compared in the literature using complete graphs, complete bipartite graphs and random graphs [8, 30, 34]. We use mainly the

same graphs as in the earlier experiments, but we have included larger complete and random graphs to the test graph set. Since CA, CA1 and CA2 behave similarly to ST for graphs without triangles, bipartite graphs are excluded from the comparison. Only ST, CA, CA1 and CA2 are run for the largest graphs. In previous works, graphs with fewer than 5000 edges have been used, while in this work the largest graph considered has 499500.

Information on the test graphs is collected in Table 4. For all graphs, we have listed the name of the graph (graph) and the number of vertices ($|V|$) and edges ($|E|$). Then the lower bound for thickness (lb) is given. If the lower bound is known to be optimal, it is marked with a star ($\star$). For graphs with unknown optimum, the lower bound is obtained by applying Euler's polyhedron formula [16].

The total number of test graphs is 47.[2] The first 21 graphs in Table 4 are complete graphs with the number of vertices varying between 10 and 1000. The next five graphs are random graphs with the number of vertices varying between 20 and 100 and the number of edges varying between 92 and 1508. Graphs r0 – r9 are random graphs with 1000 vertices and the number of edges varying between 5000 and 449500. Graphs rr1 – rr11 are random regular graphs generated with an algorithm by Steger and Wormald [36]. The degree of the vertices varies between 10 and 500.

5.3 Comparison of ST, CA, CA1 and CA2 for thickness

The total number of test graphs for the fast algorithms was 47. The best solutions for the fast heuristics and the greedy heuristics are listed in Table 4.

CA1 and CA2 outperformed ST and CA by a clear margin. CA1 and CA2 found the best solution for 39 and 27 graphs and the average ranks were 1.17 and 1.43 respectively. CA and ST respectively found only once and twice tied best solutions and their average ranks were 2.87 and 3.91. These comparison results are collected in Table 5. The reason for the relative performance of CA2 and CA1 seems to be that CA2 adds many triangles with a common edge, and therefore it constructs planar graphs with large degree. This means that the vertices that are added later get smaller degree. For large regular graphs, it seems to be a better strategy to extract subgraphs that are as regular as possible. CA1 found better solutions for large complete graphs (K_{600} – K_{1000}) and dense random graphs (r3 – r9 and rr6 – rr11). For complete graphs with fewer than 600 vertices and sparse random graphs (r1 – r2 and rr0 – rr5), CA2 obtained approximations at least as good as CA1.

The running time difference of the heuristics is clear. The relative order of the algorithms from the slowest to the fastest is CA, CA1, CA2 and ST. The explanation for the relative order of CA and CA1 (CA2) is that CA1 (CA2) extracts larger planar subgraphs and therefore the number of edges in the remaining graph decreases faster. See Figure 6 for the average running times in seconds of

[2]The random graphs can be downloaded from `http://http://www.cs.uta.fi/~tp/apptopinv` (April 27, 2006). The other graphs can be constructed by giving the command line parameters for *apptopinv* [31].

Table 5: Comparison of the performance of ST, CA, CA1 and CA2 for thickness.

	ST	CA	CA1	CA2
Number of times heuristic is the best or tied for the best	1	2	39	27
Average rank	3.91	2.87	1.17	1.43

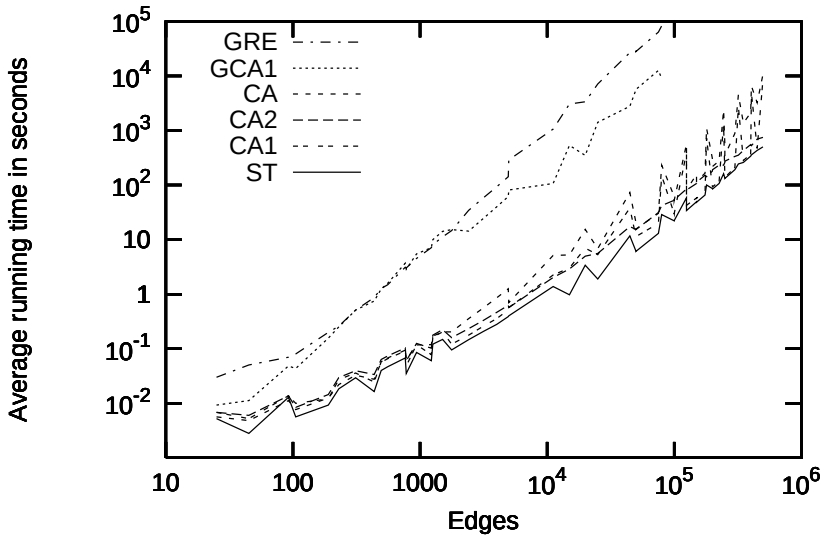

Figure 6: Average running times of ST, CA, CA1, CA2, GCA1 and GRE for thickness. Notice that the axes are logarithmic.

CA, CA1, CA2 and ST as a function of the number of the edges. The sharp turns in the curves are due to the influence of the random test graphs. The running time is higher for a random graph containing the same number of edges than for a complete graph. The average running times of GRE and GCA1 are drawn to illustrate the running time differences of the heuristics.

5.4 Comparison of GRE, GCA, GCA1 and GCA2 for thickness

The greedy algorithms achieved significantly better approximations than their non-greedy variants. For example, GCA1 decreased the solutions of CA1 to 30 percentages: CA1 got a solution of 27 for K_{90}, but the GCA1 solution was only 19. The average improvements were about 15 – 20 percentages.

GCA1 and GCA2 respectively found 24 and 21 times a best solution whereas GCA and GRE respectively found a best solution only 9 and 6 times. The average

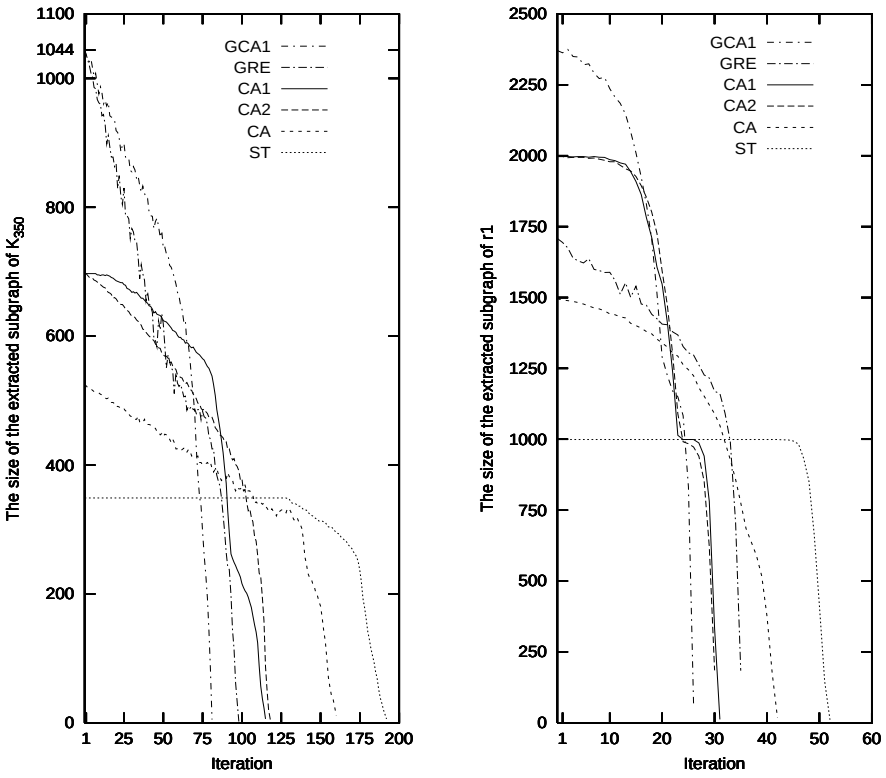

Figure 7: Sizes of the extracted planar subgraphs as a function of the number of iterations for all heuristics. In the left there are traces for K_{350} and in the right for r1.

Table 6: Comparison of the performance of GRE, GCA, GCA1 and GCA2 for thickness.

	GRE	GCA	GCA1	GCA2
Number of times heuristic is the best or tied for the best	6	9	24	21
Average rank	3.27	2.23	1.08	1.19

ranks for the heuristics were 3.27 for GRE, 2.23 for GCA, 1.08 for GCA1 and 1.19 for GCA2. These statistics are collected in Table 6.

The running times of the heuristics were very close to that of GRE, although GCA1 ran slightly faster in the case of the largest graphs. The average running times of GCA1 and GRE are illustrated in Figure 6. The running time differences between the greedy and fast heuristics are considerable: GCA1 runs 10 to 100

times slower than CA1.

There were graphs for which CA1 and CA2 outperformed GRE. CA1 and CA2 found better approximations for graphs r1, rr2, rr3 and rr4. These results are not very reliable, since for these graphs GRE was run only once. One explanation for the better performance of CA1 and CA2 against GRE for large random graphs is that the sizes of the extracted planar subgraphs are larger than they are with GRE. This is illustrated in Figure 7, where there are sample traces of ST, CA, CA1, CA2, GRE and GCA1 for a complete graph with 350 vertices (not included in the test graph set) and for r1. For K_{350}, GCA1 found the best solution and GRE found the second best solution. The extracted planar subgraphs are of maximum size in the beginning of the runs for both heuristics, but the number of edges in the extracted subgraphs of GRE decreases more rapidly than that of GCA1. The sizes of the extracted subgraphs of CA, CA1 and CA2 do not vary as much as with GRE and GCA1. The solutions of CA1 and CA2 are of the same quality, and the solutions of ST are slightly poorer. For the random graph r1 in the figure on the right, CA1 and CA2 extracts in the beginning much larger planar subgraphs than GRE. Now CA1 and CA2 yielded better approximations. The solutions of CA are worse than the solutions of GRE, but clearly better than the solutions of ST. The sizes of the extracted planar subgraphs of GCA1 were much larger at the beginning than those of CA1 and CA2 and the final solution of GCA1 was the best.

6 Conclusions

We presented two new approximation algorithms, CA1 and CA2, for the maximum planar subgraph problem and showed that the performance ratio of both algorithms is at least 7/18. The new algorithms run in linear time for bounded-degree graphs. We conjectured that the performance ratio of CA1 and CA2 is at least 4/9. All experiments performed support the conjecture. A clear goal for future research is to solve the performance ratios of CA1 and CA2. Moreover, the status of the relative performance of the better triangular cactus algorithm by Călinescu et al. [6] and the new algorithms is open.

Călinescu et al. applied their triangular cactus approach for approximating weighted MPS and MOPS. Whether our new algorithms are applicable for the weighted case remains an open question.

Acknowledgements

The author thanks the anonymous referees for their valuable comments.

References

[1] Alekseev, V.B. and Gonchakov, V.S. Thickness for arbitrary complete graphs. *Mat. Sbornik.*, 143:212–230, 1976.

[2] Beineke, L.W. and Harary, F. The thickness of the complete graph. *Can. J. Math*, 17:850–859, 1965.

[3] Beineke, L.W., Harary, F., and Moon, J.W. On the thickness of the complete bipartite graphs. *Proc. Camb. Phil. Soc.*, 60:1–5, 1964.

[4] Beyer, T., Jones, W., and Mitchell, S. Linear algorithms for isomorphism of maximal outerplanar graphs. *J. ACM*, 26(4):603–610, 1979.

[5] Călinescu, G., Fernandes, C.G., Finkler, U., and Karloff, H. A better approximation algorithm for finding planar subgraphs. *J. Algorithms*, 27(2):269–302, 1998.

[6] Călinescu, G., Fernandes, C.G., Karloff, H., and Zelikovsky, A. A new approximation algorithm for finding heavy planar subgraphs. *Algorithmica*, 36(2):179–205, 2003.

[7] Cimikowski, R. An analysis of heuristics for the maximum planar subgraph problem. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 322–331, 1995.

[8] Cimikowski, R. On heuristics for determining the thickness of a graph. *Info. Sci.*, 85:87–98, 1995.

[9] Cimikowski, R. An analysis of heuristics for graph planarization. *J. Inf. Opt. Sci.*, 18(1):49–73, 1997.

[10] Cimikowski, R. and Coppersmith, D. The sizes of maximal planar, outerplanar, and bipartite planar subgraphs. *Discr. Math.*, 149:303–309, 1996.

[11] Gabow, H.N. and Stallmann, M. Efficient algorithms for graphic matroid intersections and parity. In *Automata, Languages and Programming: 12th Colloquium*, volume 194 of *Lecture Notes in Computer Science*, pages 210–220, 1985.

[12] Golden, B.L. and Stewart, W.R. Empirical analysis of heuristics. In Lawler, E.L. and Lenstra, J.K., editors, *The Traveling Salesman Problem*, pages 207–249. John Wiley & Sons, 1985.

[13] Goldschmidt, O. and Takvorian, A. An efficient graph planarization two-phase heuristic. *Networks*, 24:69–73, 1994.

[14] Guy, R.K. and Nowakowski, R.J. The outerthickness and outercoarseness of graphs I. The complete graph & the n-cube. In Bodendiek, R. and Henns, R., editors, *Topics in Combinatorics and Graph Theory: Essays in Honour of Gerhard Ringel*, pages 297–310. Physica-Verlag, 1990.

[15] Guy, R.K. and Nowakowski, R.J. The outerthickness and outercoarseness of graphs II. The complete bipartite graph. In Bodendiek, R., editor, *Contemporary Methods in Graph Theory*, pages 313–322. B.I. Wissenchaftsverlag, 1990.

[16] Harary, F. *Graph Theory.* Addison-Wesley, 1971.

[17] Hasan, M. and Osman, I.H. Local search algorithms for the maximal planar layout problem. *Int. Trans. Oper. Res.*, 2(1):89–106, 1995.

[18] Hopcroft, J. and Tarjan, R.E. Efficient planarity testing. *J. ACM*, 21:549–568, 1974.

[19] Jayakumar, R., Thulasiraman, K., and Swamy, M.N.S. $O(n^2)$ algorithms for graph planarization. *IEEE Trans. Comp.-Aided Design*, 8(3):257–267, 1989.

[20] Jünger, M. and Mutzel, P. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16:33–59, 1996.

[21] Kant, G. An $O(n^2)$ maximal planarization algorithm based on PQ-trees. Technical Report RUU-CS-92-03, Utrecht University, Department of Computer Science, 1992.

[22] Kant, G. Augmenting outerplanar graphs. *J. Algorithms*, 21:1–25, 1996.

[23] Kleinert, M. Die Dicke des n-dimensionale Würfel-Graphen. *J. Comb. Theory*, 3:10–15, 1967.

[24] LEDA version 4.3 (commercial). Available at `http://www.algorithmic-solutions.com`.

[25] Liebers, A. Planarizing graphs - a survey and annotated bibliography. *JGAA*, 5(1):1–74, 2001.

[26] Liu, P.C. and Geldmacher, R. On the deletion of nonplanar edges of a graph. In *Proceedings of the 10th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 727–738, 1977.

[27] Lovász, L. and Plummer, M.D. *Matching Theory.* Elsevier, 1986.

[28] Mansfield, A. Determining the thickness of graphs is NP-hard. *Math. Proc. Camb. Phil. Soc.*, 93:9–23, 1983.

[29] Mitchell, S.L. Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Inf. Proc. Lett.*, 9(5):177–189, 1979.

[30] Mutzel, P., Odenthal, T., and Scharbrodt, M. The thickness of graphs: a survey. *Graphs Comb.*, 14:59–73, 1998.

[31] Poranen, T. Apptopinv - user's guide. Technical Report A-2003-3, University of Tampere, Department of Computer Sciences, 2003.

[32] Poranen, T. *Approximation Algorithms for Some Topological Invariants of Graphs*. PhD thesis, University of Tampere, 2004.

[33] Poranen, T. Heuristics for the maximum outerplanar subgraph problem. *J. Heuristics*, 11:59–88, 2005.

[34] Poranen, T. A simulated annealing algorithm for determining the thickness of a graph. *Info. Sci.*, 172:155–172, 2005.

[35] Resende, M.G.C. and Ribeiro, C.C. A GRASP for graph planarization. *Networks*, 29:173–189, 1997.

[36] Steger, A. and Wormald, N.C. Generating random regular graphs quickly. *Comb. Probab. Comput.*, 8:377–396, 1999.

[37] Tamassia, R., Di Battista, G., and Batini, C. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, 18(1):61–79, 1988.

[38] Yannakakis, M. Node- and edge-deletion NP-complete problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 253–264, 1978.

Received 10th January 2005