Experiments on Optimally Solving NP-complete Problems on Planar Graphs*

Jochen Alber Frederic Dorn Rolf Niedermeier
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany
{alber,dorn,niedermr}@informatik.uni-tuebingen.de

Abstract

Many NP-complete problems on planar graphs are "fixed parameter tractable." There are new, tree decomposition based fixed parameter algorithms solving, for example, Vertex Cover or Dominating Set optimally in time $O(c^{\sqrt{k}}n)$. Here, c is some constant, n is the number of graph vertices, and k is the size of the vertex cover or dominating set, respectively. Although these algorithms suffer from large constants c derived in the worst case analysis, the experiments in this paper show that their average case behavior is much better. For example, we were able to compute vertex covers of random planar graphs for values of $n = 3000$ and $k \approx 1300$ within few minutes. Our corresponding software package implementing the above algorithms might turn into a practically useful tool.

1 Introduction

To solve hard problems optimally, parameterized complexity proposes a "pay for what you get" approach [3, 10, 11]. Here, the point is that when searching for optimal solutions of, say Vertex Cover, one allows for algorithms that are exponential with respect to the size k of the desired vertex cover set and polynomial with respect to the input instance itself. A typical running time of such a "fixed parameter algorithm" thus is of the form $O(c^k n^{O(1)})$, where n is the input size (e.g., number of graph vertices) and c is some constant.

For many NP-complete problems on planar graphs, recently it could be shown that running times of the form $O(c^{\sqrt{k}}n)$ are achievable [1, 2]. This was the first time in parameterized complexity that a running time sublinearly growing in the exponent was proven. Although interesting from a "structural (parameterized complexity) point of view," these algorithms suffer from bad *worst case* constants c. For instance, in the case of Vertex Cover we have $c = 2^{4\sqrt{3}}$ and for Dominating Set we even have $c = 2^{12\sqrt{34}}$. On the positive side, one appealing feature of these algorithms is that they were developed in a quite general framework based on tree decompositions of planar graphs [2], which is (mathematically) more elegant and less problem-specific than "usual" fixed parameter algorithms are—particularly, those based on bounded search trees. Here, we present first experimental studies on the practical behavior of these "$c^{\sqrt{k}}$-algorithms." The main message from our studies is that they behave much better than could be expected from the theoretical worst case analysis given in [1, 2]. For example, it turns out that on a sample set of randomly generated planar graphs (with 3000 vertices) we can find optimal vertex covers of average size 1300 in average time of 19 minutes. According to the theoretical

*Supported by the Deutsche Forschungsgemeinschaft (DFG), research project PEAL (Parameterized complexity and Exact ALgorithms), NI 369/1-1.

worst case upper bounds [2] this would have taken around $9 \cdot 10^{62}$ years. The software package based on LEDA [13] we developed (and still are developing) for this purpose might turn into a useful tool for practical applications.

2 Theory and Algorithms

Tree decompositions. The key notion of this work is that of a tree decomposition of a graph. Many graph problems can be solved "easily" on trees. Hence, intuitively speaking, it is appealing to try to detect tree-like structures in graphs in order to attack hard graph problems. The notions of tree decomposition and treewidth formalize how tree-like a graph is.

Definition 1 *Let $G = (V, E)$ be a graph. A* tree decomposition *of G is a pair $\mathfrak{X} = \langle \{X_i \mid i \in I\}, T\rangle$, where each X_i is a subset of V, called a* bag, *and T is a tree with the elements of I as nodes. The following three properties must hold:*

1. *$\bigcup_{i \in I} X_i = V$;*
2. *for every edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$;*
3. *for all $i, j, k \in I$, if j lies on the path between i and k in T, then $X_i \cap X_k \subseteq X_j$.*

The width $\mathrm{tw}(\mathfrak{X})$ *of $\mathfrak{X}$ equals* $\max\{|X_i| \mid i \in I\} - 1$. *The* treewidth $\mathrm{tw}(G)$ *of G is the minimum k such that G has a tree decomposition of width k.*

To solve a graph problem using tree decompositions, one usually proceeds as follows:

Phase I: Find a tree decomposition of bounded width of the input graph,

Phase II: Solve the problem using dynamic programming on the tree decomposition (see [7]).

Fixed parameter algorithms on planar graphs. We study planar graphs, i.e., graphs that can be drawn in the plane without edge crossings. Then, (G, ϕ) will denote a plane graph, i.e., a planar graph G together with an embedding ϕ in the plane. In recent work [1, 2], a framework was developed that describes the construction of tree decomposition-based algorithms for a large class of NP-complete problems on planar graphs. More precisely, for the ease of presentation here focusing on the problems of VERTEX COVER[1] and DOMINATING SET[2] on planar graphs, the goal was to obtain "efficient" fixed parameter algorithms that solve these problems *optimally.* Obviously, since these problems are NP-complete, we have to accept exponential running times. It could be shown, however, that for parameter k being (an upper bound on) the size of the vertex cover or dominating set we search for, and n being the number of graph vertices, the running time achievable by an algorithm that executes the two phases mentioned above has a sublinear exponent in k [1, 2]:

Theorem 2 VERTEX COVER *on planar graphs can be solved in time $O(c_1^{\sqrt{k}} n)$ and* DOMINATING SET *on planar graphs can be solved in time $O(c_2^{\sqrt{k}} n)$, where $c_1 = 2^{4\sqrt{3}}$ and $c_2 = 2^{12\sqrt{34}}$.* □

[1] Given an undirected graph $G = (V, E)$ and a positive integer k, find a subset $V' \subseteq V$ of at most k vertices such that each edge in E has at least one of its endpoints in V'.

[2] Given an undirected graph $G = (V, E)$ and a positive integer k, find a subset $V' \subseteq V$ of at most k vertices such that every vertex of G either belongs to V' or has a neighbor in V' (or both).

On the positive side, this means that "one only pays for what one gets," i.e., the smaller the set we are searching for, the faster we can find it. On the negative side, clearly, the given *worst case* constants c_1 and c_2 are far from being practical. One of the main motivations for this work is to get experimental insight into how far the worst case might be from the average case.

More on constructing tree decompositions (phase I). In [2], a general methodology was developed how to, given planar graph problems such as VERTEX COVER or DOMINATING SET with parameter k, construct tree decompositions of width $O(\sqrt{k})$. To do this, one has to separate the graph in a particular, "layerwise" way. The key to this is the so-called "Layerwise Separation Property," which holds for many graph problems (see [2] for details). Here, the term "layer" refers to the following graph decomposition:

Definition 3 *Let $(G = (V, E), \phi)$ be a plane graph. The* layer decomposition *of (G, ϕ) is a disjoint partition of the vertex set V into sets $L_1, \ldots, L_r$ (called the* layers*), which are recursively defined as follows: L_1 is the set of vertices on the exterior face of G, and L_i is the set of vertices on the exterior face of $G[V - \bigcup_{j=1}^{i-1} L_j]$ for $i = 2, \ldots r$. The (uniquely defined) number r of different layers is called the* outerplanarity *of (G, ϕ), denoted by* $\mathrm{out}(G, \phi) := r$.

Informally speaking, the partitioning of the given planar graph into various chunks using separators can be chosen according to two opposing criteria, which can be "tuned" through a so-called *tradeoff-parameter:* Either one is heading for small separators (and large chunks, i.e., consisting of many layers), or one is heading for small chunks (i.e., few layers and large separators). After having separated the graph G layerwisely by separators $S_1, \ldots, S_\ell$ (each of size bounded by $O(\sqrt{k})$) into its chunks $G_0, \ldots, G_{\ell+1}$ (each of outerplanarity $O(\sqrt{k})$), one constructs tree decompositions $\mathfrak{X}_i$ for the graphs G_i, $0 \leq i \leq \ell + 1$ using an algorithm described in [1, 8]. Finally, the decompositions $\mathfrak{X}_i$ are "melted" into a (global) tree decomposition for G.

More on dynamic programming on tree decompositions (phase II). It is common knowledge that, once given a tree decomposition for a graph, many otherwise hard graph problems can be solved easily. More precisely, for treewidth ℓ phase II can be done in time $O(c^\ell N)$, where N is the number of bags of the tree decomposition and c is some constant. In case of VERTEX COVER, it is not hard to see that $c = 2$. In case of DOMINATING SET, however, this is more complicated. Telle and Proskurowski [14, 15] showed that phase II can be done in time $O(9^\ell N)$, which was recently improved to $O(4^\ell N)$ [1, 4], a significant improvement that some of the experiments in this paper owe their success.

Baker's work on approximation algorithms. In her well-known work [6], Baker gave efficient polynomial time approximation schemes for various NP-complete problems on planar graphs. In fact, already much earlier than [2] she exhibited a technique called dynamic programming on r-outerplanar graphs. For example, she showed that INDEPENDENT SET (or, equivalently, VERTEX COVER) on r-outerplanar graphs can be solved in time $O(8^r n)$. She used this to get an approximation algorithm that finds an independent set in planar graphs whose size is at least $r/(r + 1)$ optimal (see [6] for details). By way of contrast, the software package presented here is based on algorithms [1, 2] that, given the problem parameter k (i.e., the size of the solution we are searching for), solve the problems optimally for *arbitrary* planar graphs with running time "moderately" exponential with respect to k. For finding optimal solutions, hence, this approach is more general and also seems "less problem-specific" than Baker's. It is not clear to us how Baker's approach works in detail (and, in particular, what the exponential running time will be) when applied to DOMINATING SET, which appears to be a "conceptually much harder" problem than INDEPENDENT SET is.[3]

[3] Baker [6] focuses on INDEPENDENT SET and gives little technical details concerning DOMINATING SET.

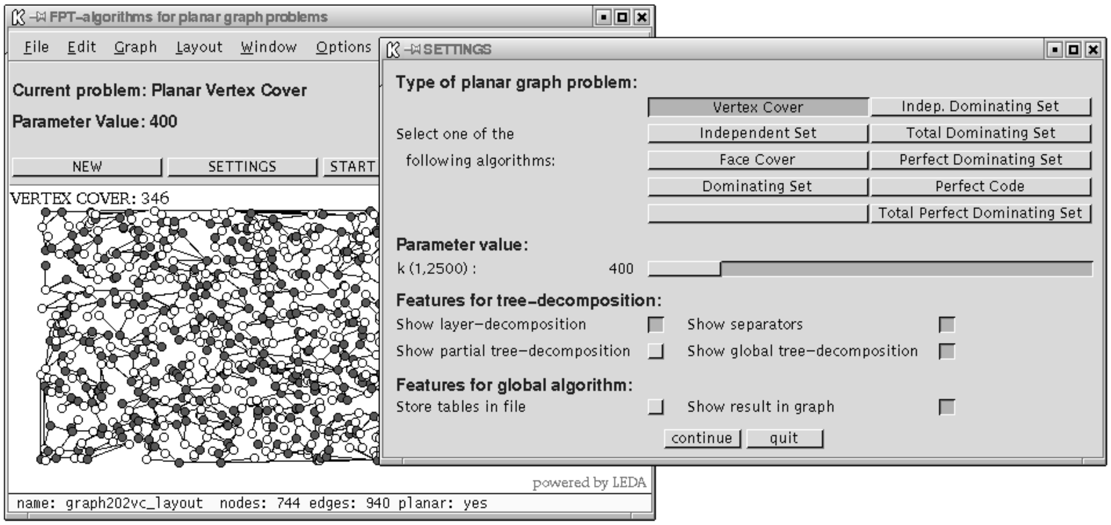


Figure 1: Software package for planar graph problems.

3 Design, Use, and Implementation

Design and Use. Based on LEDA [13], we implemented a software package which is designed for solving NP-hard problems on planar graphs optimally. More precisely, this package offers algorithms for parameterized graph problems that fit into the framework of [1, 2], i.e., that have the so-called "Layerwise Separation Property." These include Vertex Cover, Independent Set, Dominating Set, Face Cover and variations thereof, such as Total Dominating Set or Perfect Code.

The usage of the package is fairly easy. We provide a panel for the various settings of the algorithm (see Fig. 1 for a screenshot). The user selects the type of problem that (s)he wants to solve and chooses the parameter value k (i.e., the size of the desired vertex cover, independent set, dominating set, etc. the algorithm is checking for). Besides, some extra optional features of the output can be adjusted. For example, it is possible to trace the algorithm by asking for outputs of the layer decomposition, the "layerwise" separators, the tree decomposition, or the tables of the dynamic programming in various extra windows.

The planar input graph can be drawn either directly or may be given as a file in `GraphWin gw`-format. The algorithm then solves the problem, i.e., it outputs that there is no optimal solution of size at most/at least k (for minimization problems) or it computes an optimal solution and highlightens the set of vertices corresponding to it.

Implementation. So far, our software package has been implemented by a graduate student within nine months, producing around 5000 lines of C++ code based on the LEDA package [13] ((non-commercial) version 4.2). The underlying machine is a conventional 750 MHz LINUX PC with 720 MB main memory. The implementation still has to be called a prototype—numerous future fine-tuning improvements are foreseeable. It is done in an object-oriented way and offers, among others, the following main procedures. Concerning the tree decomposition phase, we have, e.g.,

- `layer_decomposition(GraphWin& gw)`, creating a layer decomposition of a plane graph given in an object `GraphWin& gw` of LEDA,
- `separator(GraphWin& gw)`, determining "layerwise separation" of a plane graph, which depends on the parameter value k and the considered graph problem, and

- `tree_decomp(GraphWin& gw)`, creating a tree decomposition for all subgraphs generated by the graph separators and melting them into a global tree decomposition of the plane graph.

Concerning the dynamic programming phase, we implemented a class `TD` for tree decompositions with various subroutines, e.g.,

- `TD.make_nice()`, transforming a tree decomposition into a "nice" [7, 8] one;
- `TD.reduce()`, reducing the size of the tree decomposition by combining two neighboring bags, whenever they appear to be subsets one of another;
- `TD.two_coloring()` and `TD.three_coloring()`, respectively, creating tables (corresponding to the bags in the tree decomposition) for VERTEX COVER (where two values are sufficient) and DOMINATING SET (where three values are necessary), respectively;[4] and
- `TD.compare_tables_VC()` and `TD.compare_tables_DS()`, being the main procedures in the dynamic programming part for VERTEX COVER and DOMINATING SET and determining optimal solutions of the given graph problem by comparing table (i.e., bag) values "bottom-up" in the tree decomposition tree, see [1, 4] for details.

To conclude this section, let us emphasize that, although the LEDA package made many things much easier, the implementation work was fairly challenging, needing several new algorithmic ideas of problems not considered in the underlying theoretical papers. Thus, special attention had to be given, e.g., to the determination of the layer of a vertex in the layer decomposition of a plane graph, to the replacement of graph vertices of degree more than three by paths of vertices of degree at most three (which is necessary for constructing a tree decomposition of small width), and to a bit-level representation of colorings to avoid waste of memory.

4 Experimental Results

In this section, we report on the experimental results obtained by running our software package on various random input samples.

4.1 Generating Random Graphs

We created a set of sample graphs using the LEDA [13] standard function `random_planar_graph (graph& G, int n, int m)` for generating (combinatorial) random planar graphs. Here, n and m (with $m \leq 3n-6$) specify the number of vertices and edges of the graph. The function, in a first step, generates a random maximal planar graph in an inductive way: For $n=3$, as an induction base, a triangle is created. For $n>3$, a random maximal planar graph of order $n-1$ is generated, an additional vertex v is added to a random face f, and all edges from v to the boundary of f are drawn. In a second step, the function `random_planar_graph` removes all but m edges at random.

Using this function, we created sample sets PGn (PG is short for "planar graphs") of random planar graphs with n vertices (where $n=$ 100, 500, 750, 1000, 1500, 2000, 3000, 4000). Here, for each graph in PGn, we chose m as a random number in the interval $[n-1, 3n-6]$. All graphs were given together with a "straight line embedding" that was computed using the LEDA standard

[4] The colors for the graph vertices encode statements as "being in the vertex cover set" or "not being in the vertex cover set." Things are more complicated for DOMINATING SET, refer to [1, 2] for details.

sample set	PG100	PG500	PG750	PG1000	PG1500	PG2000	PG3000	PG4000
size of sample	100	100	100	100	100	50	50	20
# vertices	100	500	750	1000	1500	2000	3000	4000
# edges	201.50	974.64	1483.70	1978.90	2992.00	4016.83	5895.47	8577.56
# layers	3.92	5.12	5.36	5.61	5.84	6.14	6.24	6.89
max. degree	23.19	50.79	61.23	73.31	90.57	103.77	123.10	156.61
avg. degree	4.03	3.90	3.96	3.96	3.99	4.02	3.93	4.29
treewidth (by phase I)	6.99	9.33	10.32	11.11	12.24	13.26	13.67	15.17
avg. bagsize	4.20	4.22	4.33	4.37	4.46	4.67	4.46	4.86
variance of bagsize	2.51	3.59	4.15	4.33	5.02	5.30	5.49	6.41
# bags	75.68	389.90	583.19	779.62	1167.20	1538.76	2345.96	3054.06
depth of tree	19.09	52.77	70.63	82.53	115.22	136.56	173.37	242.11
max. degree in tree	7.04	29.13	37.02	54.25	66.27	79.86	158.80	151.39
time (sec): phase I	0.62	8.11	26.90	48.96	153.08	351.05	900.30	2486.09
size of subsample (VC)	100	100	100	99	98	47	44	14
time (sec): phase II (VC)	0.06	5.41	12.35	34.46	116.40	192.17	402.36	1015.91
total time (sec): VC	0.68	13.52	39.24	82.83	265.34	503.76	1063.82	3099.89
size of VC	47.23	225.17	341.97	453.21	683.00	924.31	1326.54	1827.79
size of subsample (DS)	92	59	35	29	10	2	2	0
time (sec): phase II (DS)	16.00	208.70	340.50	302.17	936.76	1635.97	1810.48	–
total time (sec): DS	17.91	228.49	438.77	396.33	1022.88	1684.99	1975.57	–
size of DS	24.90	155.78	246.77	373.76	532.00	825.50	1182.00	–

Table 1: Summary of experimental results. The numbers in the various rows are taken as the average over graphs in PGn of the corresponding column. The abbreviations in the first column are explained in detail in Subsections 4.1, 4.2, and 4.3. Each block of rows refers to one Subsection.

function `int straight_line_embedding()`. Various graph structural data for the sample sets PGn is given in the first block of rows in Table 1.

- *size of sample:* number of random planar graphs in sample PGn;
- *# vertices:* average number of vertices of graphs in PGn;
- *# edges:* average number of edges of graphs in PGn;
- *# layers:* average number of layers of graphs in PGn using the standard straight-line embedding offered by LEDA;
- *max. degree:* average maximum degree of graphs in PGn;
- *avg. degree:* average "average degree" of graphs in PGn;

As outlined in Section 2, the algorithms proceed in two phases. In phase I, a tree decomposition of the given graph is constructed. Phase II then solves the given problem by a dynamic programming approach on the given tree decomposition.

4.2 Phase I: Constructing Tree Decompositions

Phase I of the algorithm, i.e., the construction of a tree decomposition of the given plane graph, is common to all problems offered by our software package.
Evaluation. In this phase, we measured the following figures for a given random input graph:

- *treewidth:* width of tree decomposition obtained;
- *avg. bagsize:* average size of the bags of the tree decomposition;
- *variance of bagsize:* variance of the size of the bags of the tree decomposition;

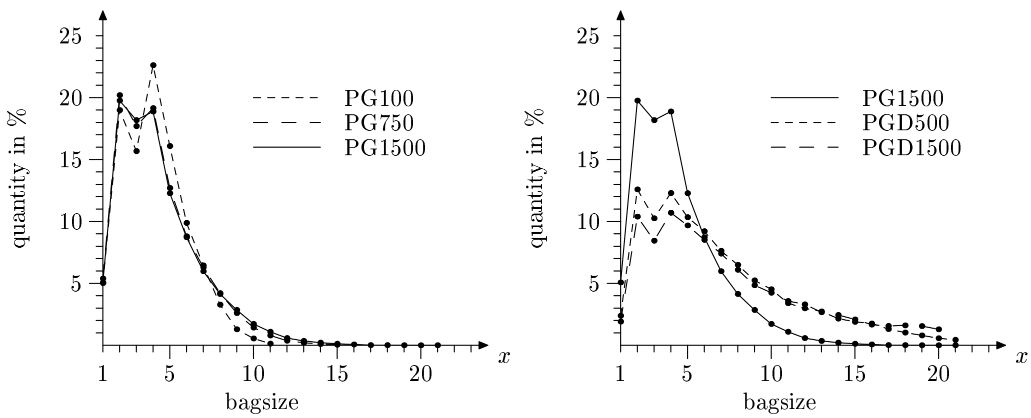


Figure 2: Distribution of sizes of bags in tree decompositions obtained by phase I.

- *# bags:* number of bags of tree decomposition;
- *depth of tree:* depth of the tree in tree decompositions;
- *max. degree in tree:* maximum degree of the tree in tree decompositions;
- *time phase I:* time needed to perform phase I of the algorithm (in seconds).

The second block of rows in Table 1 summarizes the corresponding values averaged over each sample set PGn individually. In addition, for each input graph, we investigated the distribution of the various sizes of different bags that appeared in the tree decomposition. More precisely, for each graph G and the tree decomposition $\mathfrak{X}$ obtained, we explored the percentage of bags having size s (where $s \in [1, tw(\mathfrak{X}) + 1]$). This distribution is fundamental for the running time of phase II of the algorithm. The left-hand diagram in Fig. 2 shows this distribution averaged over the graphs of selected sample sets PGn.

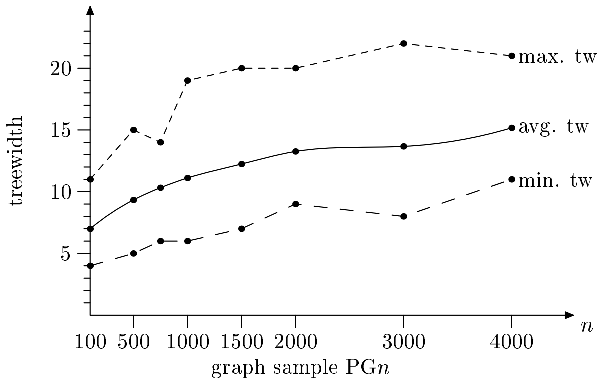
Discussion. We comment on several aspects of this part of the algorithm. Note that the treewidth obtained by phase I (see left-hand diagram of Fig. 3) is much below the worst case upper bound derived in [2]. Namely, the upper bound given there was (in terms of the size of a minimum vertex cover)

$$\mathrm{tw}(G) \quad \leq \quad 4\sqrt{3\,\mathrm{vc}(G)},$$

where $\mathrm{vc}(G)$ denotes the size of a minimum vertex cover of the given graph G. As an example, take the sample PG750. Here, the average size of a minimum vertex cover is 341.97 (see row "size of VC" in Table 1), the average treewidth obtained by phase I, however, is 10.32 which is by a *factor* 12.41 (sic!) less than the worst case upper bound value $4\sqrt{3 \cdot 341.97} \approx 128.12$. This comparison yields an even more drastic gap between worst case upper bound and practice in terms of Dominating Set. Here, the upper bound given in [1, 2] is $\mathrm{tw}(G) \leq 6\sqrt{34\,\mathrm{ds}(G)}$, where $\mathrm{ds}(G)$ denotes the size of a minimum dominating set of the given graph G.

Note that in our setting the random input graphs, in general, turned out to have few layers only. As a consequence, in order to construct a tree decomposition, the algorithm is not forced to firstly execute the step of "layerwisely" separating the graph (see the description of the algorithm in Section 2 and [2] for details). Hence, it directly uses the algorithm for graphs of bounded outerplanarity (see the long version of [1] or [8, Theorem 83]). Still, it is remarkable, that the worst case upper bound $\mathrm{tw}(G) \leq 3\,\mathrm{out}(G, \phi) - 1$, for a plane graph $(G\phi)$ with outerplanarity $\mathrm{out}(G, \phi)$ is too pessimistic. Take again, e.g., the sample PG750 where the average treewidth is 10.32, which is by a factor 1.46 lower than $3 \cdot 5.36 - 1 \approx 15.08$.

The most remarkable observation is the *structure* of the tree decompositions obtained by this phase of the algorithm, namely the distribution of the sizes of the bags and of the number of



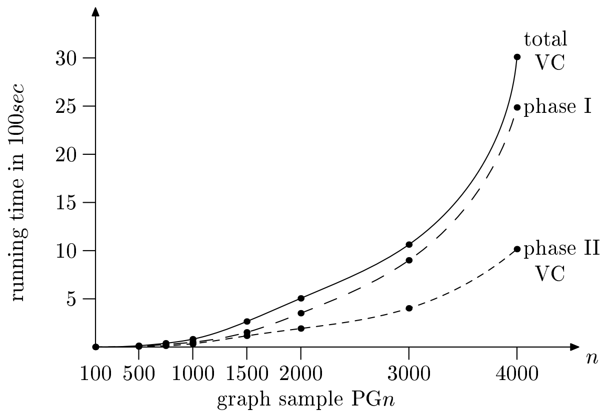


Figure 3: Treewidth and running time. Left diagram illustrates treewidth values obtained by phase I for the graph samples PGn. Right diagram illustrates the total running time of our algorithm for Vertex Cover split into the two phases of the algorithm.

nodes in the tree. As to the first point, it is interesting that the average size of the bags in the tree decompositions in *all* of the samples PGn is around 4.5 (seemingly independent of the width of the tree decompositions and the size of the input graphs). Also, the variance of the bagsize distribution is very low. To illustrate this, consider Fig. 2, which shows this distribution for various sample sets PGn. Note that the size of a very high percentage of the bags is in the range of 1–6, and only few bags are of big size (determining the width of the tree decomposition).

Also, the number of nodes in the tree decomposition is quite low. Recall that the algorithm (see the long version of [1] or [8, Theorem 83]) for constructing the tree decomposition $\mathfrak{X}$ of a given graph G is based on finding "in a layerwise fashion" a suitable spanning tree T_S for G (more precisely for a supergraph $\hat{G}$ of degree less than four). Then, the tree T of $\mathfrak{X}$ basically is given by a subdivision[5] of T_S. Hence, for an n-vertex graph G we would expect at least $2n-1$ bags in $\mathfrak{X}$. However, the tree decompositions in average turn out to have only around $0.75n$ many bags. This improvement by a factor of 2.67 over the expected number of bags is due to the `TD.reduce()` function (see Section 3). Both, the distribution of the bagsizes and the number of bags have a direct influence on the running time of phase II of the algorithm.

4.3 Phase II: Dynamic Programming on Tree Decompositions

The second phase of the algorithm is a problem-dependent dynamic programming approach. Since our current software package does not (yet) implement the ideas of [5] to reduce memory requirements,[6] we performed this second phase only on a subset of our original graph samples. In the case of Vertex Cover (where the space requirement to store all tables is $O(2^{\text{tw}}N)$, if tw is the treewidth of the underlying tree decomposition and N is the number of its bags), we restricted ourselves to those graphs for which phase I yields a tree decomposition of width at most 17; in the case of Dominating Set (where the space requirement is $O(4^{\text{tw}}N)$), we only considered graphs with decompositions of width at most 9.

Evaluation. For this part of the algorithm, the following figures were measured:

- *size of subsample* (VC/ DS)*:* number of graphs in PGn that were considered in phase II, i.e., that had treewidth at most 17 in the case of Vertex Cover (VC) and at most 9 in the case of Dominating Set (DS).

[5] Additional nodes are put on every edge of T_S.

[6] In our current version, all tables for the dynamic programming are kept in main memory at the same time.

- *time phase II* (VC/ DS)*:* time (in seconds) needed to perform phase II of the algorithm in the case of Vertex Cover/Dominating Set on the sample subset;
- *total time* (VC/ DS)*:* time (in seconds) needed to perform phase I and phase II of the algorithm in the case of Vertex Cover/Dominating Set on the sample subset;
- *size of* VC/ DS*:* size of minimum Vertex Cover/ Dominating Set of the sample subset.

Discussion. The time needed to perform the dynamic programming can be determined by considering the distribution of the bagsizes of the tree decomposition obtained by phase I (see Subsection 4.2). More precisely, if, for a given tree decomposition $\mathfrak{X}$ with N nodes, $\text{bag}(s)$ (where $s \in [1, tw(\mathfrak{X})+1]$) denotes the number of bags having size s, then the running time for phase II basically is $\sum_{s=1}^{tw(\mathfrak{X})+1} \text{bag}(s) \cdot c^s$. Here, c is a constant depending on the problem (for Vertex Cover, $c = 2$; and for dominating set, $c = 4$ [4]) Now, clearly, both, the distribution "bag" and the value $N = \sum_{s=1}^{tw(\mathfrak{X})+1} \text{bag}(s)$ do have direct influence on this running time.

The reasonably short running times in our experiments are due to the structure of the tree decompositions obtained in phase I. Note that we found vertex covers of average size 1326.54 in PG3000 in less than 19 minutes average time. Compare this running time with the running time of $8.70 \cdot 10^{62}$ years, which is given by the worst case upper bound from Theorem 2, when plugging in $k = 1326.54$, $n = 3000$ and when assuming a machine with 10^9 operations per second. To put it the other way round, assuming the same type of machine the worst case upper bound from Theorem 2 suggests that in 19 minutes we can compute a vertex cover of size 16.92 only (compare this figure with the average size 1326.54 that we actually compute).

4.4 Alternative Random Graphs

Besides the sample sets PGn that were created in a purely combinatorial way using the LEDA function `void random_planar_graph(graph& G,int n,int m)`, we also dealt with a modified random graph generation. The key motivation here was to generate sample sets of graphs with more layers than those in PGn. We achieved this by generating sample sets PGDn (short for "planar graphs Delaunay"). In contrast to PGn, a different, purely geometric procedure was used in order to yield a maximal planar graph of order n. A vertex set of size n was randomly placed in the plane, and then some Delaunay triangulation was computed, from which edges were removed at random. The corresponding embedding (inherited by the Delaunay triangulation) indeed proved to have more layers and, hence, both the absolute treewidth and the average size of the bags of the decomposition obtained by phase I were higher. Also, the distribution of the bags tended to have higher percentage of large bags (see the right-hand diagram in Fig. 2 for a comparison to the sample sets PGn).

Finally, we remark that test sets created by other geometric generating schemes (for example, using maximal planar graphs that were triangulated by a sweeping algorithm instead of a Delaunay triangulation) resulted in similar findings than those of the sample sets PGn.

5 Conclusion

Implementing and experimenting with fixed parameter algorithms for NP-complete problems on planar graphs, we discovered first pieces of circumstantial evidence that practical problem instances of these problems might be solved in an efficient way optimally. We also reinforced grounds for the practical significance of the concept of tree decompositions of graphs (see [12] for an up-to-date account on heuristic approaches for constructing tree decompositions).

Among others, we detected optimal vertex covers in planar graphs of sizes more than $k = 1000$ within few minutes. This might be compared with recent experimental results of Dehne *et al.* [9], where search tree algorithms for VERTEX COVER on general graphs were parallelized on 10 Sun SPARC workstations. They optimally solved VERTEX COVER instances for $k \approx 400$ within a minutes range.

Many things remain to be done. Our software package should be improved and extended by, e.g., (1) experimenting with different planar embeddings to optimize phase I, (2) making the construction of the tree decomposition more efficient (using heuristics) and also trying to further lower bag and thus table sizes, (3) reducing the memory requirement for the tables in the dynamic programming, maybe using ideas from [5], and (4) further easing the use of the software, e.g., by also providing meta-information such as expected remaining running time during the execution.

References

[1] J. Alber, H. L. Bodlaender, H. Fernau, and R. Niedermeier. Fixed parameter algorithms for planar dominating set and related problems. In *Proceedings 7th SWAT*, Springer-Verlag LNCS 1851, pp. 97–110, 2000. Long version accepted for publication in *Algorithmica*.

[2] J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speed-up for planar graph problems. In *Proceedings 28th ICALP*, Springer-Verlag LNCS 2076, pp. 261–272, 2001. Long version available as Technical Report TR01-023, Electronic Colloquium on Computational Complexity (ECCC), Trier, March 2001.

[3] J. Alber, J. Gramm, and R. Niedermeier. Faster exact solutions for hard problems: a parameterized point of view. *Discrete Mathematics*, 229:3–27, 2001.

[4] J. Alber and R. Niedermeier. Improved tree decomposition based algorithms for domination-like problems. Manuscript, September 2001. Submitted for publication.

[5] B. Aspvall, A. Proskurowski, and J. A.Telle. Memory requirements for table computations in partial k-tree algorithms. *Algorithmica*, 27:382–394, 2000.

[6] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41:153–180, 1994.

[7] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proceedings 22nd MFCS'97*, Springer-Verlag LNCS 1295, pp. 19–36, 1997.

[8] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.

[9] F. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. Solving large FPT problems on coarse grained parallel machines. Manuscript, July 2001.

[10] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[11] M. R. Fellows. Parameterized complexity: new developments and research frontiers. In *Aspects of Complexity*, De Gruyter. To appear 2001.

[12] A. M. C. A. Koster, H. L. Bodlaender, and S. P. M. Hoesel. Treewidth: computational experiments. *Electronic Notes in Discrete Mathematics 8*, Elsevier Science Publishers, 2001.

[13] K. Mehlhorn and S. Näher. *LEDA: A Platform of Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, England, 1999.

[14] J. A. Telle and A. Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In *Proc. 3rd WADS*, Springer-Verlag LNCS 709, pp. 610–621, 1993.

[15] J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, 10(4):529–550, 1997.