Discrete Applied Mathematics, Vol. 145(2), pp. 219-231, 2005

Experimental Evaluation of a Tree Decomposition Based Algorithm for Vertex Cover on Planar Graphs [1]

Jochen Alber Frederic Dorn Rolf Niedermeier

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Fed. Rep. of Germany

E-mail: {alber,dorn,niedermr}@informatik.uni-tuebingen.de

Abstract

Many NP-complete problems on planar graphs are "fixed-parameter tractable:" Recent theoretical work provided tree decomposition based fixed-parameter algorithms exactly solving various parameterized problems on planar graphs, among others Vertex Cover, in time $O(c^{\sqrt{k}}n)$. Here, c is some constant depending on the graph problem to be solved, n is the number of graph vertices, and k is the problem parameter (for Vertex Cover this is the size of the vertex cover).

In this paper, we present an experimental study for such tree decomposition based algorithms focusing on Vertex Cover. We demonstrate that the tree decomposition based approach provides a valuable way of exactly solving Vertex Cover on planar graphs. Doing so, we also demonstrate the impressive power of the so-called Nemhauser/Trotter theorem which provides a Vertex Cover-specific, extremely useful data reduction through polynomial time preprocessing. Altogether, this underpins the practical importance of the underlying theory.

1 Introduction

Since Robertson and Seymour [25] defined the notion of tree decomposition together with the associated graph parameter treewidth, this played an important role in algorithm theory. Many in general NP-complete graph problems do have polynomial or even linear time solving algorithms when the underlying graph has a tree decomposition of width bounded by a constant. To compute

[1] Supported by the Deutsche Forschungsgemeinschaft (DFG), research project PEAL (Parameterized complexity and Exact ALgorithms), NI 369/1-1,1-2.

the treewidth of a graph, however, is an NP-hard problem. For constant ℓ, linear time algorithms to determine whether a graph has treewidth ℓ have been developed, but due to huge constants in their running times they are still of little practical use (see [10,11,22] for a survey).

In this work, we deal with the question of how NP-hard problems on planar graphs can be solved exactly in practice using tree decompositions. Despite the above mentioned facts, by applying an in a way more problem-specific approach, we obtain encouraging empirical results. To keep the presentation within reasonable size, we concentrate on the NP-complete Vertex Cover [2] problem on planar graphs. Vertex Cover is among the most popular and most important problems in combinatorial optimization [15]. In our recent theoretical work [2], we showed that computing an optimal vertex cover of size at most k (if existent) can be done in time $O(c^{\sqrt{k}}n)$, where $c = 2^{4\sqrt{3}}$. (Similar results with larger constants c hold for Independent Set, Dominating Set and related problems [1,2].) We present an empirical evaluation of the given algorithm and, additionally, we study the influence of a clever, Vertex Cover-specific data reduction through preprocessing based on a theorem of Nemhauser and Trotter [24]. Even though further heuristic improvements would be foreseeable when attacking practical problems, we restricted our empirical analysis to algorithmic techniques that have provable performance bounds. Our findings are that the average-case behavior is much better than what is indicated by the worst-case bounds given in the theoretical analysis. More importantly, the treewidths of the considered combinatorial random planar graphs, as a rule, were much below the values predicted by the theoretically analyzed bounds.

Vertex Cover on general graphs has seen considerable interest in recent parameterized complexity studies. To solve hard problems such as Vertex Cover optimally, parameterized complexity proposes a "pay for what you get" approach [3,17,18]. Here, the point is that when searching for optimal solutions of 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. Using a search tree algorithm with many case distinctions, Vertex Cover on general graphs can now be solved in time $O(1.29^k + kn)$ [14]. Notably, in case of planar graphs no significantly improved search tree algorithms seem available. By way of contrast, using the concept of tree decompositions, running time $O(c^{\sqrt{k}}n)$ is achievable [1,2] which is a significant asymptotic improvement. This result

[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 each edge in E has at least one of its endpoints in V'.

is obtained by showing that a planar graph having a vertex cover of size at most k must have treewidth $O(\sqrt{k})$ (see Section 2 for details). Actually, in our experiments with combinatorial random graphs it turned out that only a special case of the tree decomposition based algorithm was appropriate. In this special case, the parameter value k played no role for the construction of the tree decomposition[3] but the construction was governed by the number of layers of the given planar graph. As a result, we could directly solve the optimization version of the problem. We can easily construct simple examples, however, where the parameter k becomes dominant in the construction of the tree decomposition (see Section 2 for details). Our empirical studies reveal that the tree decomposition based algorithm is competitive and the theoretical worst-case bound on running time might be much too high. For example, it turned out that on a sample set of randomly generated planar graphs (with 3000 vertices) we can find optimal vertex covers of average size 1375 in average time less than one minute using a conventional 750 MHz LINUX PC with 720 MB main memory. Our investigations are based on a software package that (in its final version) will implement exact solving algorithms for many hard problems on planar graphs. It is based on LEDA [23] and might turn into a useful tool for practical applications. The part designed for Vertex Cover is freely available [32].

2 From a Theorist's Point of View

Tree Decompositions. The key notion of this work is that of a tree decomposition of a graph. The notions of tree decomposition and treewidth formalize how tree-like a graph is (see [10,11] for surveys).

Definition 1 Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $\mathcal{X} = \langle \{X_i \mid i \in I\}, T\rangle$ where each X_i is a subset of V, called *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 $i, j, k \in I$, if j lies on the path between i and k in T, then $X_i \cap X_k \subseteq X_j$.

The *width* $\mathrm{tw}(\mathcal{X})$ of $\mathcal{X}$ 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.

[3] Observe that, as a rule of thumb, minimum vertex cover sets of the planar graphs in our setting usually contain up to one half of all graph vertices. Hence, it appears to be favorable to trade exponential growth with respect to parameter k for exponential growth with respect to the number of layers of the planar graph.

Tree decomposition based algorithms usually proceed 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 [10] and [4,30,31] for more specific results).

In the case of Vertex Cover, as observed by Chen *et al.* [14], there exists a powerful data reduction technique due to Nemhauser and Trotter [24]. Hence, to solve Vertex Cover we additionally have a ***Phase 0*** that performs a preprocessing on the given input graph. This results in a (size) reduced graph, the so-called "problem kernel." Then, Phases I and II solely work on the problem kernel.

Fixed-Parameter Algorithms and 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. Our goal is to obtain "efficient" fixed-parameter algorithms that provide *optimal* solutions. 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 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 *Let k denote an upper bound on the size of a minimum vertex cover of a graph G. Then, for the treewidth of a planar graph G, we have $\mathrm{tw}(G) \leq 4\sqrt{3k} + 5$. A corresponding tree decomposition can be constructed in time $O(\sqrt{k}n)$. Moreover,* Vertex Cover *on planar graphs can be solved in time $O(c^{\sqrt{k}}n)$ with $c = 2^{4\sqrt{3}}$.* $\square$

In particular, if $\mathrm{vc}(G)$ denotes the size of an optimal vertex cover of a graph G then we also obtain $\mathrm{tw}(G) \leq 4\sqrt{3\,\mathrm{vc}(G)} + 5$ for the planar case. This upper bound is optimal up to constant factors as can be seen from the complete $n \times n$-grid graph which has treewidth $\Theta(n)$ and vertex cover size $\Theta(n^2)$. On the positive side, Theorem 2 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* constant c is far too big in order to make this algorithmic approach looking practical.

More on Phase 0. There is a well-known, simple reduction to problem kernel for Vertex Cover attributed to Buss [17]. The idea is that if we are looking for a vertex cover of size at most k, then a vertex with degree $k + 1$ *has* to be part of such a vertex cover because, otherwise, k vertices would not suffice to cover all edges emanating from v. The benefits of this data reduction applied to planar graphs are limited: as a rule of thumb, vertex covers of the planar graphs in our setting consist of approximately half of all vertices (see Section 4), whereas usually no vertex of degree $\geq \mathrm{vc}(G)$ exists. Hence, we concentrate on a more involved and more powerful data reduction technique. Chen *et al.* [14] made the following important observation on a linear size problem kernel for Vertex Cover (which even holds for general graphs). It is based on a theorem of Nemhauser and Trotter [24], an alternative, constructive proof of which appears in [8] (also see [21] for a very recent account on this result).

Theorem 3 *There is an algorithm of running time $O(kn + k^3)$ that given an instance (G, k) for* Vertex Cover *constructs another instance (G', k') where $k' \leq k$ and G' contains at most $2k'$ vertices such that G has a vertex cover of size k iff G' has a vertex cover of size k'.* $\square$

Together with Theorem 2, this results in a time $O(2^{4\sqrt{3k}}k + kn)$ algorithm for Vertex Cover on planar graphs. Actually, Theorem 3 determines a set of vertices in G which have to be part of a minimum vertex cover, and leaves a graph G' where the remaining "vertex cover vertices" of G have to be found. The underlying algorithm computes a maximum matching in a bipartite graph. In this way, Theorem 3 can be applied in a polynomial time preprocessing phase to reduce the original input instance G to a smaller instance G'. In a sense, G' then bears the "really hard" part of the Vertex Cover instance, the *problem kernel*. As we will see in the experimental evaluation, this data reduction through preprocessing allows impressive improvements concerning the width of the tree decompositions to be constructed as well as enormous speed-ups of the whole algorithm.

By way of contrast to Theorem 2, in a non-constructive manner Alon *et al.* [5] obtained the more general result that if a graph G excludes a K_h minor, then $\mathrm{tw}(G) \leq h^{3/2}\sqrt{n}$. Using the linear problem kernelization above we obtain a graph G' with n' vertices and the guarantee that $n' \leq 2\,\mathrm{vc}(G')$. Specializing to planar graphs where $h = 5$, this yields the treewidth bound $\mathrm{tw}(G') \leq 5^{3/2}\sqrt{2\,\mathrm{vc}(G')} \approx 15.8\sqrt{\mathrm{vc}(G')}$, whereas our constructive bound from Theorem 2 is approximatley $6.9\sqrt{\mathrm{vc}(G')}$.

More on 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 4 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 $\mathrm{out}(G, \phi) := r$.

Specifically, we compared Theorem 2 with the following result [11] (also see [1]) for graphs of bounded outerplanarity. This result is also used for proving Theorem 2 by applying it to "graph chunks" of outerplanarity $O(\sqrt{k})$.

Theorem 5 *For an r-outerplanar connected graph $G = (V, E)$ (that is given together with its corresponding embedding in the plane) a tree decomposition $\langle \{X_i \mid i \in I\}, T\rangle$ of width at most $3r - 1$ and with $2n - 1$ tree nodes can be found in time $O(rn)$.* $\square$

Combining Theorems 2 and 5, we directly obtain the following.

Corollary 6 *Given a planar graph G and its embedding ϕ in the plane that has a vertex cover of size at most k, a tree decomposition of width*

$$\min\{3 \cdot \mathrm{out}(G, \phi) - 1,\ 4\sqrt{3k} + 5\}$$

can be efficiently computed. $\square$

Corollary 6 allows us to choose between two different strategies for constructing tree decompositions. Our experiments with combinatorial random graphs showed that the tree decomposition based on Theorem 5 always was preferable because graphs as considered here always turned out to have not too many layers such that the method behind Theorem 2 always appeared to be inferior in this setting. It would become beneficial when dealing with plane graphs with high outerplanarity and still reasonable vertex cover sizes, two somewhat opposing criteria. In Fig. 1, however, we give an easy example of a graph containing a number of layers linearly related to the vertex cover size. In this situation the approach behind Theorem 2 would be preferable.

We briefly describe the construction behind Theorem 2. Informally speaking, the partitioning of the given planar graph into various chunks using a set of 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 $G_1, \ldots, G_{\ell+1}$ (each of outerplanarity $O(\sqrt{k})$), one constructs tree decompositions $\mathcal{X}_i$ for the graphs G_i, $1 \le i \le \ell + 1$ using

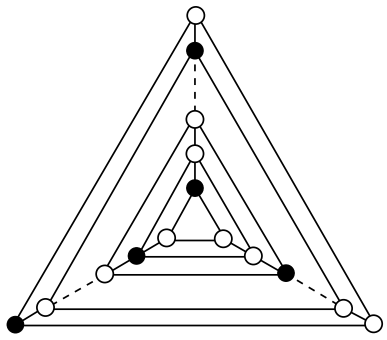


Fig. 1. Example for a linear dependence between the number of layers and the size of a vertex cover. Suppose that the graph has r layers. Then the size of an optimal vertex cover is $\mathrm{vc}(G) = 2r$. (An optimal vertex cover is given by the white vertices.) In this setting, we get $4\sqrt{3\,\mathrm{vc}(G)} + 5 \leq 3r - 1$ for $r \geq 15$.

the algorithm in Theorem 5. Since each graph chunk G_i has at most $O(\sqrt{k})$ many layers we know that $\mathrm{tw}(\mathcal{X}_i) = O(\sqrt{k})$. Finally, the decompositions $\mathcal{X}_i$ are "melted" into a (global) tree decomposition for G with the help of the separators $S_1, \ldots, S_\ell$. This can be done in a way such that the (global) tree decomposition again has width at most $O(\sqrt{k})$.

The aforementioned tradeoff-parameter allows us to tune this algorithm in two opposing directions. For a large value of the tradeoff-parameter, the algorithm allows for large separators, but only graph chunks with few layers. Conversely, for a small value of the tradeoff-parameter, the algorithm only uses small separators, but admits graph chunks with many layers. In the extremal case (i.e., when the tradeoff-parameter is set to zero), no separation is done and the graph is considered as a single "big" chunk which is processed by the algorithm in Theorem 5.

From a theoretical point of view, there is an optimal choice of the tradeoff-parameter which guarantees that the width of the resulting tree decomposition is bounded by $4\sqrt{3k} + 5$ (see [2] for details). From a practical point of view, in our implementation, the adjusting of the tradeoff-parameter is left to the user and can be chosen in an application-specific manner.

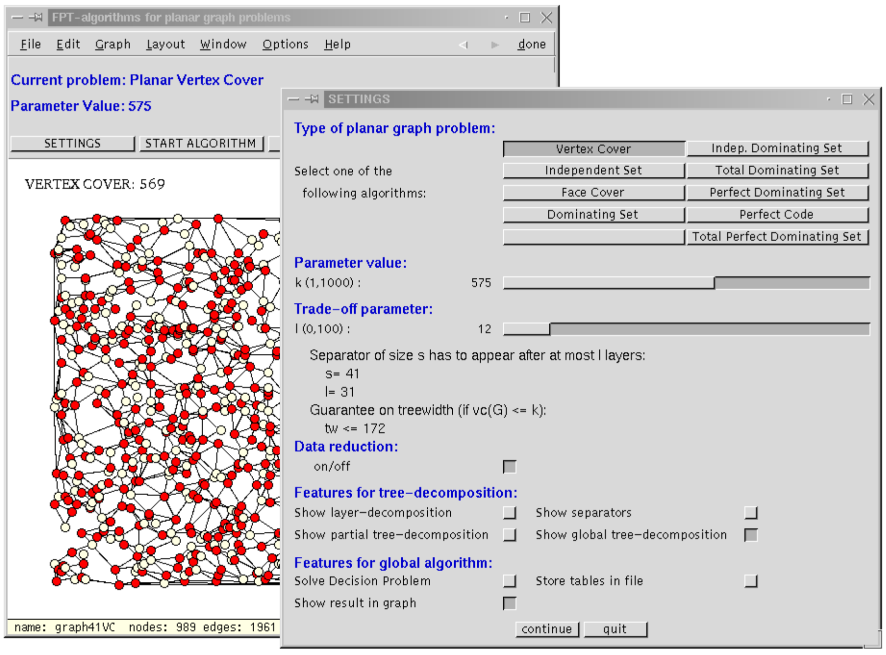
More on Phase II. It is common knowledge that, once given a tree decomposition for a graph, many otherwise hard graph problems can be solved easily. More precisely, for treewidth ℓ, Phase II can be done in time $O(d^\ell N)$, where N is the number of bags of the tree decomposition and d is some constant. For this phase, one typically uses a dynamic programming approach. In case of Vertex Cover, it is not hard to see that $d = 2$. The idea is as follows. Check for all of the $|I|$ many bags X_i, $i \in I$, each time, all $2^{|X_i|}$ vertex cover candidate sets for the subgraph $G[X_i]$ of G induced by the vertices from bag X_i. This information is stored in tables A_i ($i \in I$). In a second step, these tables are compared against each other. Each bag of the tree decomposition thus has a table associated with it. The comparison process works in a bottom-up fashion from the leaves to the root of the tree decomposition, comparing "neighboring" tables (whose corresponding tree nodes are connected by an edge) against each other and updating the current information. During this updating process it is guaranteed that the "local" solutions for each subgraph associated with a bag of the tree decomposition are combined into a "global," optimal solution for the overall graph G by bookkeeping the various best possible vertex cover sizes (cf., e.g., [1,4]).

Baker's Work on Approximation Algorithms. In her influential work [7], 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 independent sets in planar graphs whose sizes are at least $r/(r+1)$ optimal (see [7] for details). In a sense, the exact part of Baker's result for graphs of bounded outerplanarity is intrinsically supported as a special case by our software package: If Phase I of our algorithm does not separate the graph in a layerwise fashion at all, the guarantee on the treewidth for an r-outerplanar graph is $3r-1$, which, together with Phase II, results in a time $O(2^{3r-1}n) = O(8^r n)$ algorithm for Vertex Cover. Observe, however, that the difference still is that we did use a tree decomposition based approach as sketched in [1,11].

3 From a User's Point of View

Design and Use. Based on LEDA [23], we implemented a software package which is designed for exactly solving NP-hard problems on planar graphs. 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 Independent Dominating Set, Total Dominating Set, or Perfect Code. At the time being, the implementation concerning Vertex Cover is the one that achieved the highest level of maturity.

The usage of the package is fairly easy. We provide a panel for the various settings of the algorithm (see Fig. 2 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). In addition, we leave it to the user to adjust



```
Current problem: Planar Vertex Cover
Parameter Value: 575

VERTEX COVER: 569

name: graph41VC   nodes: 989 edges: 1961

Type of planar graph problem:
Select one of the
  following algorithms:
Parameter value:
k (1,1000) :          575
Trade-off parameter:
l (0,100) :           12
   Separator of size s has to appear after at most l layers:
      s= 41
      l= 31
   Guarantee on treewidth (if vc(G) <= k):
      tw <= 172
Data reduction:
   on/off
Features for tree-decomposition:
Show layer-decomposition          Show separators
Show partial tree-decomposition   Show global tree-decomposition
Features for global algorithm:
Solve Decision Problem            Store tables in file
Show result in graph
```

Fig. 2. Software package for hard planar graph problems.

the tradeoff-parameter (described in the previous section). Besides, some extra optional features for 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. A restriction of this package to Vertex Cover is freely available [32].

The planar input graph can be drawn either directly or may be given as a file. The algorithm then solves the problem, i.e., it outputs that there is no optimal solution of size at most k (for minimization problems) or at least k (for maximization problems) or it computes an optimal solution and highlightens the set of vertices corresponding to it. If, however, the tradeoff-parameter is set to zero (as will be done in our experiments) then Phase I proceeds according to the algorithm behind Theorem 5. In this setting, no-instances are not detected in Phase I and the algorithm solves the optimization version of the problem.

Implementation. Our software package consists of more than 5000 lines of C++ code based on the LEDA package [23] ((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. Although the LEDA package made many things much easier, the implementation work was fairly challenging, dealing with several algorithmic questions not considered in the underlying theoretical papers. 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 or 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). As memory quickly becomes a bottleneck, it was essential to encode the table entries of the dynamic programming as bit words, thus making it necessary to operate with "bit-masking" and dealing with bit parallelism at the word level. Another thing worth pursuing in order to save memory is to do a refined analysis of which tables have to be kept open (i.e., to be kept in main memory) during the dynamic programming process. In this context, it might be beneficial to experiment with choosing different root nodes of the tree decomposition. Finally, to do the updating of the tables efficiently, we determined those vertices that occur in both corresponding bags and sorted the table rows according to the bit settings of these common vertices. In summary, these and several more fine tunings were necessary to make the program competitive. Notably, all of these tunings are simple by themselves, but they are indispensable as a whole in order to get running times as presented in the following.

4 Experimental Results

We report on the experimental results obtained by running our software package to solve Vertex Cover on various random input samples. This should be considered as a first serious round of tests of our implemented algorithms.

4.1 Generating Random Graphs

We created a set of sample graphs using the LEDA [23] standard function

```
void 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 the 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, all but m edges are randomly removed. We remark that this method does not generate graphs according to the uniform distribution.

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). Each sample set PGn contains 100 sample graphs. 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 corresponding LEDA [23] standard function. Various graph structural data for the sample sets PGn is given in the first block of rows (see "Graph data") in Table 1. The values in the table are the averages for each sample set PGn over the following values which were measured for each graph G individually:

- *# vertices:* number of vertices of graph G;
- *# edges:* number of edges of graph G;
- *# layers:* number of layers of graph G using the standard straight-line embedding offered by LEDA;
- *max. degree:* maximum degree of graph G;
- *avg. degree:* "average degree" of graph G;
- *size of VC:* size of minimum vertex cover of graph G (as computed by our algorithm);

What parameter value k should be chosen in our setting? We argue that, for the random planar instances generated here, the parameter k (as long as we do not use unreasonable small values for k) should be exchanged with the parameter that upperbounds the number of layers: It turned out that, on the one hand, the graphs generated in our setting had few layers only using a simple straight-line embedding ϕ, i.e., the number $\mathrm{out}(G, \phi)$ is small. In nearly all cases we had less than 10 layers. On the other hand, the size of an optimal vertex cover seemed to be a large parameter: As a rule of thumb, for the graphs generated in our setting, we may say that an optimal vertex cover contained about half of the vertices of the graph. This means that even for small graphs (with 100 vertices), the parameter value for k (i.e., the size of the vertex cover we seek for) should be relatively high. Recall the upper bound $\min\{3\,\mathrm{out}(G, \phi) - 1, 4\sqrt{3k} + 5\}$ on treewidth from Corollary 6. Taking into account that we had less than 10 layers and that a reasonable choice of k should be close to the size of an optimal vertex cover, the minimum above is always taken by the first term. As a consequence, trading the parameter k for the parameter $\mathrm{out}(G, \phi)$ seems reasonable in our setting. In practice, this means that the tradeoff-parameter was set to zero for the graphs considered here such that the algorithm behind Theorem 2 omits the layerwise separation of the graph and a tree decomposition for G is computed directly using Theorem 5. Consequently, a tree decomposition will be constructed in any case and we can solve the optimization version of the problem on the given tree decomposition. In fact, our experiments showed that the widths of the tree decompositions obtained without doing separation were much smaller compared to the ones obtained after separation.

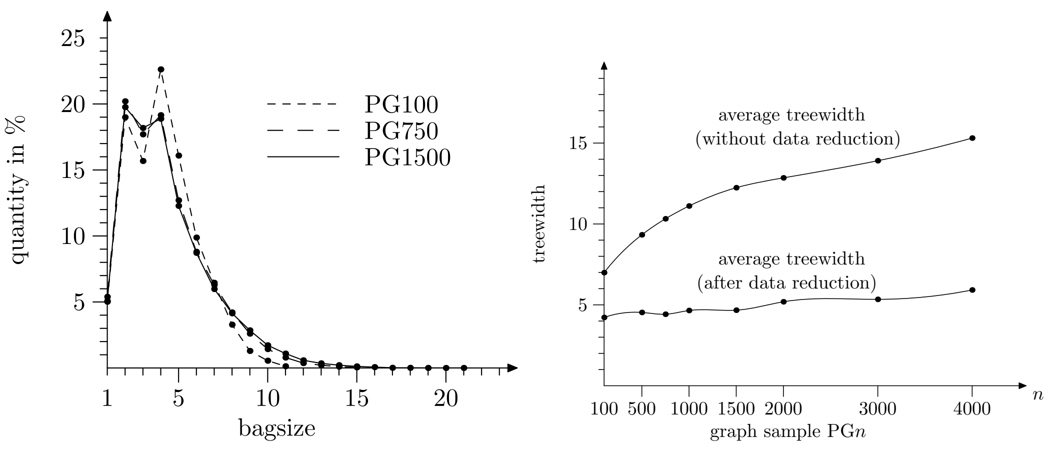


Fig. 3. Bag distribution and treewidth. The left diagram illustrates the distribution of sizes of bags in tree decompositions obtained by Phase I (without data reduction). The right diagram compares the treewidth values obtained without running preprocessing Phase 0 to the values obtained after running Phase 0.

4.2 Experiments Without Data Reduction

We ran the combination of Phases I and II in order to solve Vertex Cover on the given graphs in the sample sets PGn.

Evaluation. In the described setting, we measured the following figures for each given random input graph G individually:

- *width of tree dec.:* width of tree decomposition $\mathcal{X}$ obtained for G (by Phase 1);
- *avg. bagsize:* average size of the bags of $\mathcal{X}$;
- *variance of bagsize:* variance of the size of the bags of $\mathcal{X}$;
- *# bags:* number of bags of $\mathcal{X}$;
- *depth of tree:* depth of the tree in $\mathcal{X}$;
- *max. degree in tree:* maximum degree of the tree in $\mathcal{X}$;

The averages of these values over all graphs from a given sample set are summarized in the second block of rows (see "Tree decompositions obtained") in Table 1.

Besides, we recorded the running times for the two phases. The corresponding values can be seen in the third block of rows in Table 1 (see "Time needed").

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 $\mathcal{X}$ obtained, we explored the percentage of bags having size s (where $1 \leq s \leq \text{tw}(\mathcal{X}) + 1$). This distribution influences the running time of Phase II of the algorithm. The left-hand diagram in Fig. 3 shows this distribution averaged over the graphs

Without data reduction

sample set	PG100	PG500	PG750	PG1000	PG1500	PG2000	PG3000	PG4000

Graph data:

# vertices	100	500	750	1000	1500	2000	3000	4000
# edges	201.5	974.6	1483.7	1978.9	2992.0	3960.8	6070.6	8264.5
# layers	3.92	5.12	5.36	5.61	5.84	6.11	6.29	6.86
max. degree	23.2	50.8	61.2	73.3	90.6	104.9	129.6	146.6
avg. degree	4.03	3.90	3.96	3.96	3.99	3.96	4.05	4.13
size of VC	47.2	225.2	342.0	453.9	683.6	917.3	1373.8	1856.8

Tree decompositions obtained (by Phase I):

width of tree dec.	6.99	9.33	10.32	11.11	12.24	12.85	13.91	15.31
highest occurring width	11	15	14	19	20	20	22	21
avg. bagsize	4.20	4.22	4.33	4.37	4.46	4.57	4.62	4.72
variance of bagsize	2.51	3.59	4.15	4.33	5.02	5.12	5.61	6.26
# bags	75.7	389.9	583.2	779.6	1167.2	1552.3	2327.7	3087.1
depth of tree	19.1	52.8	70.6	82.5	115.2	122.6	163.0	196.3
max. degree in tree	7.0	29.1	37.0	54.3	66.3	89.3	150.1	185.7

Time needed:

time (sec): Phase I	0.34	2.96	6.90	12.46	30.20	48.73	128.38	253.05
time (sec): Phase II	0.06	5.41	12.35	34.46	116.40	192.17	402.36	1015.91
total time (sec):	0.40	8.37	19.25	46.92	146.60	240.90	530.74	1268.96

Table 1
Summary of experimental results for tree decomposition based algorithms without data reduction. The numbers in the various rows are taken as the *averages* over 100 graphs in PGn of the corresponding column. The abbreviations in the first column are explained in detail in Subsections 4.1 and 4.2.

of selected sample sets PGn.

Discussion. Recall from the discussion at the end of Subsection 4.1 that the worst-case upper bounds on treewidth in Theorem 5 are much smaller than the ones obtained by Theorem 2. As an example, consider the sample set PG750. Here, the average size of a minimum vertex cover is around 342 (see row "size of VC" in Table 1). For such a value, the worst-case upper bound in Theorem 2 yields $4\sqrt{3 \cdot 342} + 5 \approx 133$. Conversely, observing that the average number of layers of these graphs is 5.36, and using this value in the worst-case upper bound of Theorem 5, we expect treewidth around $3 \cdot 5.36 - 1 \approx 15$.

Consider the average treewidth obtained by the method behind Theorem 5 (see row "treewidth" in Table 1). Our main observation is that the upper bound $\mathrm{tw}(G) \leq 3\,\mathrm{out}(G, \phi) - 1$ from Theorem 5 is too pessimistic with respect to an average behavior. Take again, e.g., the sample set PG750 where the average treewidth is 10.3. This is by a factor 1.46 lower than the value 15 which could have been expected since the average number of layers was 5.36. The same holds true for the graph in PG750 which had the highest treewidth of 14. This graph, however, had 7 layers which means that the worst-case upper bound would have guaranteed a width of 20 (by a factor 1.43 worse than what we obtained).

Moreover, we made two interesting observations:

- 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). To illustrate this, consider the left-hand diagram of Fig. 3 which shows the distribution of the sizes of the bags 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 large.
- The number of nodes in the tree decomposition is lower than the value $2n-1$ given in Theorem 5. The tree decompositions in average turn out to have only around $0.75n$ many bags. This improvement by a factor of 2.7 over the expected number of bags is due to an implemented heuristic method which reduces the size of the tree decomposition by combining two neighboring bags whenever one appears to be a subset of the other.

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: Very recently, the notion of *f-cost* was introduced as a more refined measure for the quality of a tree decomposition [12]. Here, $f : \mathbb{N} \to \mathbb{R}^+$ is some function and the f-cost of a tree decomposition $\mathcal{X} = \langle \{X_i \mid i \in I\}, T\rangle$ is defined to be $\Sigma_{i\in I} f(|X_i|)$. Since, in Phase II, the time (and space) needed to process a node of the tree decomposition whose associated bag has size k roughly is $f(k) = 2^k$ the time needed for Phase II is $f(\mathcal{X}) := \Sigma_{i\in I} f(|X_i|)$. In this sense, the distribution of the bag sizes (and the low average bag size) measured in our experiments contribute to a small f-cost and, hence, a fast running time for Phase II.

Summary. In a nutshell, the key message from the experimental studies in this subsection is that—at least on the random planar graphs used in our setting—Phases I and II perform much better than could have been expected by their worst-case analysis. The low total running time for solving Vertex Cover on these instances is mainly due to the relatively well-behaving tree decompositions (both in terms of width and structure).

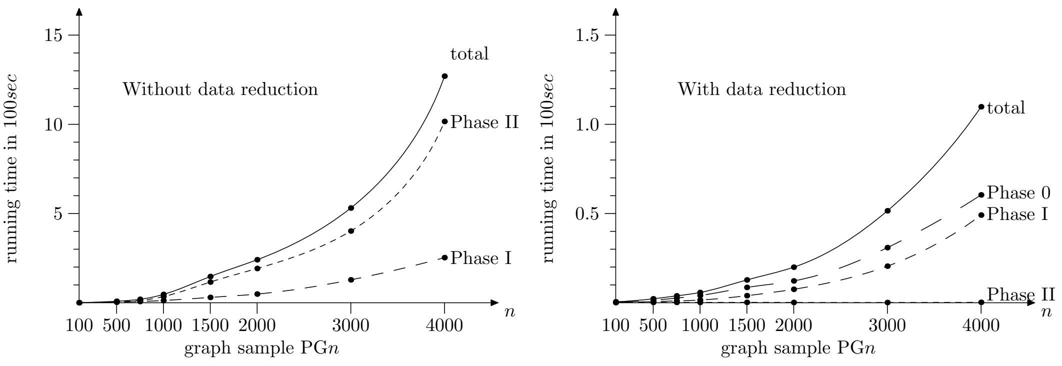


Fig. 4. Running time. The left and the right diagram, respectively, show (the various contributions for) the running times without the preprocessing (see Subsection 4.2) and with the preprocessing (see Subsection 4.3), respectively. Note that the time-axis is scaled down by a factor of 10 in the right-hand diagram.

4.3 Experiments With Data Reduction

A similar series of experiments as described in Subsection 4.2 was carried out with an additional preprocessing Phase 0 (see Section 2). The data reduction from Phase 0 reduces the size of the input instances and leaves us with reduced graph instances which bear (in the sense of finding a minimum vertex cover) the computationally challenging graph parts.

It is important to note that by applying Phase 0 we are no longer able to generate *all* minimum vertex covers. By way of contrast, this possibility is preserved when only running Phases I and II.

Evaluation. For each graph in a set PGn, we iteratively applied the data reduction due to Nemhauser and Trotter (Theorem 3)[4] until the graph could not be reduced any further. We measured the following figures:

- *# vertices removed:* the number of vertices removed by the data reduction;
- *# edges removed:* the number of edges removed by the data reduction;
- *# vertices for VC found:* the number of vertices that could be determined by the preprocessing to be in an optimal vertex cover;

The first block of rows in Table 2 (see “Data reduction”) shows these values.

After Phase 0, we are left with the reduced graphs from our sample set PGn. Graph-structural data of the reduced instances is collected in the second block of rows (see “Graph data (for remaining reduced instances)”) in Table 2.

[4] In addition, in each iteration, we first of all removed degree one vertices and chose their neighbors to be in the optimal vertex cover we seek for.

With data reduction

sample set	PG100	PG500	PG750	PG1000	PG1500	PG2000	PG3000	PG4000
Data reduction:								
# vertices removed	56.2	337.9	518.1	684.2	1054.9	1347.1	1963.9	2507.1
# edges removed	103.7	642.7	1045.0	1372.3	2166.2	2660.8	3999.3	5467.0
# vertices for VC found	23.1	133.8	209.7	273.9	429.3	544.9	783.8	1007.1
Graph data (for remaining reduced instances):								
# vertices	43.8	162.1	231.9	315.8	445.1	652.9	1036.1	1492.9
# edges	97.8	331.9	438.7	606.7	825.8	1300.0	2071.4	3110.6
# layers	2.40	2.49	2.57	2.64	2.60	2.91	2.93	3.20
max. degree	12.5	18.1	17.5	20.1	19.0	27.5	29.4	38.4
avg. degree	3.38	3.16	3.08	3.10	3.05	3.17	3.21	3.32
size of VC of red. graph	24.1	91.4	132.2	180.0	254.3	372.4	590.1	849.7
Tree decompositions obtained (for remaining reduced instances):								
width of tree dec.	4.22	4.53	4.42	4.65	4.67	5.19	5.34	5.92
highest occurring width	10	9	10	12	16	14	14	16
avg. bagsize	3.70	3.50	3.43	3.41	3.37	3.48	3.46	3.54
variance of bagsize	0.80	0.69	0.56	0.54	0.50	0.68	0.61	0.76
# bags	27.7	97.9	136.8	188.9	287.8	399.8	626.5	912.3
depth of tree	8.1	15.6	15.3	17.1	17.6	25.0	26.3	33.8
max. degree in tree	3.2	11.9	19.7	24.7	39.9	49.4	71.9	94.3
Time needed:								
time (sec): Phase 0	0.30	1.38	2.63	4.06	8.60	12.24	30.90	60.45
time (sec): Phase I	0.15	0.63	1.02	1.52	3.97	7.54	20.51	49.15
time (sec): Phase II	0.08	0.20	0.21	0.17	0.22	0.16	0.13	0.24
total time (sec):	0.53	2.21	3.86	5.75	12.79	19.93	51.54	109.84

Table 2
Summary of experimental results for tree decomposition based algorithms with data reduction as provided by Phase 0. The numbers in the various rows are taken as the *averages* over 100 graphs in PGn of the corresponding column. The abbreviations in the first column are explained in detail in Subsection 4.3.

While running Phases I and II on these reduced graph instances, we recorded, analogously to the tests in Subsection 4.2, the relevant figures for the structure of the tree decompositions (refer to the third block of rows in Table 2 ("Tree decompositions obtained")).

Finally, the running times for the various phases are given in the last block of rows of Table 2.

Discussion. The data reduction has an impressive impact on both the size of the remaining reduced graphs and the width of their tree decompositions:

- Around 60–70% of the vertices and
- around 60–70% of the edges of the original graphs

were removed by the preprocessing phase. Moreover, the preprocessing detected a very high percentage (around 50–60%) of vertices that can be guaranteed to belong to a minimum vertex cover.

The widths of the tree decompositions for the reduced graphs are considerably smaller than the widths of the tree decompositions obtained for the original graphs. As an example, again take the (reduced) graphs from PG750, where the treewidth in average now is 4.42, whereas the average width of the original graphs was 10.32, a decrease by more than 57% (see Fig. 3 for a comparison of the average treewidths obtained with and without data reduction, respectively).[5] The number of layers decreased from an average of 5.36 to 2.57. In all sample sets (seemingly independent of the size of the original graphs) we observed an average bagsize of around 3.5 for the reduced graphs.

As a result of the well-behaving tree decompositions for the reduced graph instances, we obtained a drastic improvement of the running times for Phase I and especially for Phase II. Whereas in the setting of Subsection 4.2, Phase II played a crucial role in the overall running time, the contribution of this phase is almost neglectable when running the preprocessing (see Fig. 4 for an illustration). The major part of the overall running time now is spent by the preprocessing Phase 0.

The key message is that data reduction pays off. The larger the graphs are, the better the speed-up gained by the preprocessing: Whereas we get an average speed-up by a factor of approximately 3.8 for smaller graphs from PG500, the corresponding factor is around 11.6 for the graphs in PG4000.

Summary. The experiments in this subsection revealed that—at least on the random planar graphs used here—the data reduction suggested by Nemhauser and Trotter allows significant improvements concerning the width of the tree decompositions and, thus, drastically speeds-up the overall running time for the whole algorithm. The only negative aspect herein is that no more all optimal vertex covers can be generated.

[5] This is, however, only an average behavior: The graph which originally had the highest width (of 14) in the sample set PG750 could only be brought down to treewidth 10 after Phase 0.

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 (see [23, Section 10.4] for details). The corresponding embedding (inherited by the Delaunay triangulation) indeed proved to have more layers and, due to the higher "degree of cyclicity" of graphs obtained in such a way, both the absolute treewidth and the average size of the bags of the decomposition obtained by Phase I were much higher. Also, the distribution of the bags tended to have higher percentage of large bags. The tree decompositions obtained without doing a preprocessing in many cases had too high treewidth to make vertex cover algorithmically tractable with the dynamic programming approach. The impact of the data reduction by Nemhauser and Trotter was not as considerable as in the case of 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 as those of the sample sets PGn.

5 Conclusion

Implementing and experimenting with exact algorithms for Vertex Cover on planar graphs, we discovered pieces of circumstantial evidence that practical problem instances of these problems might be solved in an efficient way exactly. We also reinforced grounds for the practical significance of the concept of tree decompositions of graphs (see [13,22] for up-to-date accounts on heuristic approaches for constructing tree decompositions). Among others, we detected optimal vertex covers in planar graphs of sizes more than $k = 1500$ (see PG4000) within less than two minutes on a standard PC. This might be compared with recent experimental results of Dehne *et al.* [16], where search tree algorithms for Vertex Cover on *general* graphs were parallelized on

10 Sun SPARC workstations. Within a time range of several minutes, they optimally solved Vertex Cover instances in parallel for parameter values k around 400.

Practical Challenges. Our software package should be improved by, e.g.,

- experimenting with different planar embeddings (other than the straight-line embedding used so far) in order to further optimize Phase I,
- making the construction of the tree decomposition more efficient (using heuristics, e.g., the ones proposed in [13]) and also trying to further lower bag and thus table sizes,
- reducing the memory requirement for the tables in the dynamic programming, perhaps using ideas from [6] or [9], and also trying to bring the number of tables kept simultaneously in main memory at a minimum, and
- further easing the use of the software, e.g., by also providing meta-information such as expected remaining running time during the execution (so-called progress indicators).

Concerning time and space efficiency, it is open to explore whether there are significant differences with respect to the decision versus the optimization version of Vertex Cover. A major issue in the area between theory and practice is the question of how to deal with the separators in constructing the tree decomposition in a more sophisticated way (in a sense, [2] handles this point relatively carelessly) such that the bag sizes and the treewidth can be kept as small as possible.

Another direction to pursue in future research is the generation of "hard" graph instances to further test the described algorithms. The work of Sanchis [28,29] might serve as a stimulus here.

Theoretical Challenges. A question raised by Robin Thomas is whether using the concept of branchwidth [26] instead of treewidth might lead to (more) efficient solutions. Notably, branchwidth of planar graphs can be computed in polynomial time [27] and there are first practical experiences with implementing an algorithm constructing planar branch decompositions [20]. A promising approach for the Dominating Set problem where the use of branch decompositions improves the previously best known tree decomposition based algorithms can be found in [19]. Another issue of concern is whether the so-called Thomas' lemma (cf. [17, Section 7.2]), yielding very "well-behaving" tree decompositions, is of use in our setting.

Acknowledgments. This work profited from constructive feedback during the workshop "Optimization Problems, Graph Classes and Width Parameters" organized by Andrzej Proskurowski and Oriol Serra at the Centre de Recerca Matemàtica, Barcelona, November 15–17, 2001, where, in particular, Robin Thomas and Ioan Todinca gave valuable comments and hints. The presentation could be significantly improved due to constructive comments of two anonymous referees of *Discrete Applied Mathematics.*

References

[1] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.

[2] J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speed-up for planar graph problems. In *Proceedings 28th ICALP 2001*, 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. In *Proceedings 5th LATIN 2002*, Springer-Verlag LNCS 2286, pp. 613–627, 2002.

[5] N. Alon, P. D. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proceedings 22nd STOC '90*, pp. 293-299, ACM, 1990.

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

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

[8] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25: 27–46, 1985.

[9] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, 8:216–235, 1987.

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

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

[12] H. L. Bodlaender and F. V. Fomin. Tree decompositions with small cost. In *Proceedings 8th SWAT 2002*, Springer-Verlag LNCS 2368, pp. 378–387, 2002.

[13] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca. On treewidth approximations. *Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW'01)*. 2001.

[14] J. Chen, I. A. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001.

[15] P. Crescenzi and V. Kann. How to find the best approximation results—a follow-up to Garey and Johnson. *ACM SIGACT News*, 29(4):90–97, 1998.

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

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

[18] M. R. Fellows. Parameterized complexity: the main ideas and some research frontiers. In *Proceedings 12th ISAAC 2001*, Springer-Verlag LNCS 2223, pp. 291–307, 2001.

[19] F. V. Fomin and D. M. Thilikos. Dominating sets in planar graphs: branch-width and exponential speed-up. To appear in *Proceedings 14th ACM-SIAM SODA*, Baltimore, MD, USA, 2003.

[20] I. V. Hicks. Planar branch decompositions. Submitted to *Discrete Applied Mathematics*, 2001.

[21] S. Khuller. Algorithms column: the vertex cover problem. *ACM SIGACT News*, 33(2):31–33, 2002.

[22] 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.

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

[24] G. L. Nemhauser and L. E. Trotter. Vertex packing: structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

[25] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[26] N. Robertson and P. D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory Series B*, 52:153–190, 1991.

[27] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[28] L. A. Sanchis. Test case construction for the vertex cover problem. *Computational Support for Discrete Mathematics*, pp. 315-326, 1994.

[29] L. A. Sanchis. Generating hard and diverse test sets for NP-hard graph problems. *Discrete Applied Mathematics*, 58:35–66, 1995.

[30] J. A. Telle and A. Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In *Proceedings 3rd WADS '93*, Springer-Verlag LNCS 709, pp. 610–621, 1993.

[31] 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.

[32] `http://www-fs.informatik.uni-tuebingen.de/peal/index.html`