Engineering Planar Separator Algorithms

MARTIN HOLZER, FRANK SCHULZ, and DOROTHEA WAGNER
KIT, Universität Karlsruhe (TH)
and
GRIGORIOS PRASINOS, and CHRISTOS ZAROLIAGIS
CTI & University of Patras

We consider classical linear-time planar separator algorithms, determining for a given planar graph a small subset of its nodes whose removal divides the graph into two components of similar size. These algorithms are based on planar separator theorems, which guarantee separators of size $O(\sqrt{n})$ and remaining components of size at most $2n/3$ (where n denotes the number of nodes in the graph). In this article, we present a comprehensive experimental study of the classical algorithms applied to a large variety of graphs, where our main goal is to find separators that do not only satisfy upper bounds, but also possess other desirable characteristics with respect to separator size and component balance. We achieve this by investigating a number of specific alternatives for the concrete implementation and fine-tuning of certain parts of the classical algorithms. It is also shown that the choice of several parameters influences the separation quality considerably. Moreover, we propose as planar separators the usage of fundamental cycles, whose size is at most twice the diameter of the graph: For graphs of small diameter, the guaranteed bound is better than the $O(\sqrt{n})$ bounds, and it turns out that this simple strategy almost always outperforms the other algorithms, even for graphs with large diameter.

Categories and Subject Descriptors: G.2.1 [**Combinatorics**]: Combinatorial Algorithms; G.2.2 [**Graph Theory**]: Graph Algorithms; G.4 [**Mathematical Software**]: Algorithm Design and Analysis; D.2.8 [**Metrics**]: Performance Measures; E.1 [**Data Structures**]: Graphs and Networks

General Terms: Algorithms, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Divide-and-conquer, planar graph, separator

This work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority – 6th FP), under contracts no. IST-2002-001907 (integrated project DELIS) and no. FP6-021235-2 (project ARRIVAL).

Authors' addresses: M. Holzer, F. Schulz, and D. Wagner, KIT, Universität Karlsruhe (TH), Institut für Theoretische Informatik, Postfach 69 80, 76128 Karlsruhe, Germany, email: {martin.holzer, dorothea.wagner}@kit.edu; frank.schulz@ptv.de. G. Prasinos and C. Zaroliagis, R.A. Computer Technology Institute, N. Kazantzaki Str, Patras University Campus, 26500 Patras, Greece, and Department of Computer Engineering and Informatics, University of Patras, 26500 Patras, Greece, email: {green,zaro}@ceid.upatras.gr.

DOI 10.1145/1498698.1571635 http://doi.acm.org/10.1145/1498698.1571635

ACM Reference Format:
Holzer, M., Schulz, F., Wagner, D., Prasinos, G., and Zaroliagis, C. 2009. Engineering planar separator algorithms. ACM J. Exp. Algor. 14, Article 1.5 (August 2009), 31 pages.
DOI = 10.1145/1498698.1571635 http://doi.acm.org/10.1145/1498698.1571635

1. INTRODUCTION

Finding node separators in planar graphs is a fundamental and well-studied problem with several applications, the most prominent of which is its employment in divide-and-conquer approaches (see, e.g., Alber et al. [2003], Frederickson [1987], Goodrich [1995], Henzinger et al. [1997], and Lipton and Tarjan [1980]). An interesting analogy to the divide-and-conquer paradigm from the realm of molecular biology is protein folding, which is the process of polypeptides (i.e., linear chains of amino acids) transforming themselves into 3D structures. In a recent paper [Ozkan et al. 2007], it is suggested that this folding takes place as a zipping-and-assembly process, where zipping denotes growing of local substructures within the polypeptide, and the assembly part refers to interaction between these substructures. This physical model has, in turn, led to more efficient algorithms for computing the stereometric structure of proteins by determining certain folding points, which are used to split up the given instance.

1.1 Problem and History

Investigation of the problem started with the pioneering work of Lipton and Tarjan [1979], who introduced the so-called Planar Separator Theorem, stating that for every planar graph with $n \geq 5$ nodes there is a subset of its nodes—called (node) separator—of size at most $\sqrt{8n} \approx 2.83\sqrt{n}$ whose removal separates the graph into two components of size smaller than $2n/3$. The proof of this theorem is constructive in the sense that an algorithm can be derived from it fairly easily, which can be implemented to run in linear time. Djidjev [1982] improved the upper bound on separator size to $\sqrt{6n} \approx 2.45\sqrt{n}$, paralleled by a refinement of Lipton and Tarjan's algorithm; he also proved a lower bound of $1.56\sqrt{n}$, which is still the best known.

Both these classical algorithms proceed in two stages, where they share the second one. The first, or simple, stage of each algorithm tries to determine a separator that meets the given bounds, induced by one or two levels of nodes of a breadth-first search (BFS) tree. If this is not possible, the algorithms enter the second, or complex, stage, which determines an appropriate fundamental cycle in a planar graph that contributes to the sought separator.

Since then, a lot of generalizations and extensions have been made. The upper bound on separator size has been improved by Alon et al. [1994] to $2.13\sqrt{n}$, and by Djidjev and Venkatesan [1997] to the currently best known bound of $1.97\sqrt{n}$—where the aforementioned core algorithm is used as a subroutine. The latter work also gives a lower bound of $1.56\sqrt{n}$ on the size of planar node separators, due to an analysis of specifically-constructed globe graphs. Spielman and Teng [1996] provided an algorithm for separation of a planar graph into two components of size, at most $3n/4$ each, through a separator of no more than

$\sqrt{3.5n} \approx 1.84\sqrt{n}$ nodes. For the sake of completeness, we mention that there are results on separating planar graphs into components of size at most $n/2$ via separators of size $\mathcal{O}(\sqrt{n})$, where the constant factors hidden in this term are much higher than those stated above.

Aleksandrov et al. [2006] consider a generalization named t-separators: Given a planar graph with cost and weight associated to each node, total cost C, and a balance factor $t \in (0, 1)$, the objective is to find a node separator with total cost not exceeding C such that after removal of the separator the weight of each remaining component is at most t times the total weight of the graph. Such separations typically yield more than two components. Setting $t = 2/3$ and using unit weights and costs, the generalized problem reduces to the original variant of Lipton and Tarjan [1979]. The article includes an experimental study with a few synthetic and real-world graphs. Finally, for the case of 2-connected planar graphs with a maximum face size of f, Miller [1984] showed that there exists a simple-cycle separator of size $2\sqrt{2fn}$ separating the graph into components of size at most $2n/3$.

1.2 Applications

Planar separator algorithms are often employed to recursively decompose planar graphs according to the divide-and-conquer paradigm. A straightforward implementation would yield a running time of $\mathcal{O}(n \log n)$. In Goodrich [1995], some refined strategies are presented to obtain a linear-time decomposition algorithm. Another application of recursive decomposition is shortest-path search on planar graphs [Frederickson 1987], where separators consisting of $\mathcal{O}(n/\sqrt{r})$ nodes and splitting the graph into $\Theta(n/r)$ components of size $\mathcal{O}(r)$ are used. This decomposition approach has been further elaborated in Henzinger et al. [1997], where, along with other techniques, a linear-time approach for the single-source shortest path problem on planar graphs is presented. Finally, Alber et al. [2003] describe parameterized algorithms for different graph problems that rely on planar separators.

1.3 Our Contribution

Although the classical algorithms by Lipton and Tarjan [1979] and Djidjev [1982] have been known for quite some time, we are not aware of any systematic experimental study investigating them. In this article, we therefore include various graphs, both synthetic and from real world, which exhibit considerable differences in properties, such as density, minimum separator size, diameter, and so on. However, we do not settle for finding separators satisfying those upper bounds, but consider several new algorithmic aspects, regarding: (i) optimization of separator size and balance; (ii) consideration of a fundamental-cycle separator (FCS) algorithm in their own right; (iii) application of postprocessing techniques to improve the quality of separators; and (iv) choices made in the implementation with respect to various parameters. Furthermore, to provide a measure for the tradeoff between separator size and balance of component sizes, we employ the separation ratio as suggested by Leighton and Rao [1999].

Our experiments show that the behavior of the tested algorithms highly depends on the input graph: For example, on regular graphs (such as grids) one level of the BFS tree computed at the first stage already is an almost-optimal separator, whereas for more irregular and real-world graphs (such as road map graphs) the classical algorithms yield relatively bad solutions.

The FCS algorithm guarantees a bound on the separator size of $2d+1$, where d denotes the diameter of a triangulation of the input graph. Thus, when the diameter is small, which is often the case for real-world graphs, this approach guarantees separators smaller than $O(\sqrt{n})$, given for the classical algorithms. A surprising outcome of our experiments is therefore that the FCS algorithm provides better solutions also for all other graphs tested.

Many subroutines shared by the classical algorithms are not fixed with respect to every detail: For the BFS performed during the simple stage, neither the root node nor the order in which incident edges and adjacent nodes are visited are specified. Further, the complex stage requires a triangulation of the graph and the selection of a nontree edge, where no further assumptions are made. Our results show the importance of an appropriate choice of the BFS root node: Since testing all nodes in the graph would incur quadratic running time, we rather propose different heuristic methods to select a set of root nodes to be considered. Even though variation of the order for visiting incident edges does not affect the distribution of the nodes to BFS levels, the "balance" of the tree edges is affected, which may have an impact on the performance of the subsequent steps. Finally, for computation of the desired triangulation, the triangulating BFS variant proposed, which simultaneously recomputes the given BFS tree, performs exceptionally well.

It should be noted that in contrast to Aleksandrov et al. [2006], which also features an experimental study on planar separator algorithms applied to both synthetic and real-world inputs, the main focus of our article is on the analysis of the degrees of freedom inherent to the classical algorithms, so as to exploit concrete ways of implementing these for improvement of separation quality in practice. A preliminary version of this work appeared in Holzer et al. [2005].

1.4 Structure

The remainder of this article is organized as follows. In the rest of this section, we discuss previous work done in the field of graph separation. Section 2 briefly reviews the Planar Separator Theorem and gives details on the two classical as well as the FCS algorithm; for our experiments, we provide several optimization criteria, and show how to modify the algorithms accordingly. In Section 3, we present some of the degrees of freedom inherent to the algorithms and propose several variants of implementing each of them; we round off this part with a description of some postprocessing techniques. Section 4 lists the classes of graphs used in our experimental study, while the experimental setup and results are reported in Section 5. In Section 6, we conclude with a short summary and some open questions on the subject.

2. ALGORITHMS

In this section, we consider classical linear-time planar separator algorithms implementing the Planar Separator Theorem, as stated below. The node separators computed by the different algorithms fulfill different upper bounds $\beta\sqrt{n}$ on the separator size, for certain constants β, while each of the remaining components contains no more than two thirds of all nodes. The first theorem of this kind (for $\beta = \sqrt{8}$) and the fundamental-cycle lemma were introduced by Lipton and Tarjan [1979]; this lemma is used in the complex stage of the classical algorithms, and further constitutes, as a stand-alone procedure, the fundamental-cycle separation algorithm. For our experiments, we envision three optimization criteria, which partly lead to some algorithmic modifications; in particular, implementing the optimized version of the complex stage requires a little elaboration.

For simplicity, we state the Planar Separator Theorem and its related algorithms for the case of an unweighted planar graph. Nevertheless, the algorithms and our implementations work equally for the weighted case, as introduced in Lipton and Tarjan [1979].

THEOREM 1 (PLANAR-SEPARATOR THEOREM). *Let $G = (V, E)$ be a planar graph with $n \geq 5$ nodes. Then, the nodes of V can be partitioned into three sets A, B, and S such that no edge joins a node in A with a node in B, neither A nor B consists of more than $2n/3$ nodes, and S contains at most $\beta\sqrt{n}$ nodes, for some constant β.*

Using this notation, the separator bound due to Lipton and Tarjan is given by $\beta = \sqrt{8}$, while Djidjev's bound takes $\beta = \sqrt{6}$. However, our implementation of Lipton and Tarjan's algorithm was done according to a textbook version with $\beta = 4$ [Mehlhorn 1984; Kozen 1992].

An important concept used in the theorem are fundamental cycles: Given a spanning tree of the input graph, a fundamental cycle consists of a nontree edge e together with the path connecting the two endpoints of e in the spanning tree.

LEMMA 1 (FUNDAMENTAL-CYCLE LEMMA). *Let G be a connected planar graph. Suppose G has a spanning tree of height h. Then, the nodes of G can be partitioned into three sets A, B, and C such that no edge joins a node in A with a node in B, neither A nor B consist of more than $2n/3$ nodes, and C is a fundamental cycle containing no more than $2h + 1$ nodes.*

Before proceeding with the algorithms, we state the following definition, which will be used later.

Definition 1 (*Radius, Diameter*). Given a (uniformly) weighted graph $G = (V, E)$, the *eccentricity* of a node $v \in V$ is defined to be the maximum distance from v to any other node in G. The minimum eccentricity over all nodes in the graph is called *radius*, while the maximum eccentricity is called *diameter*.

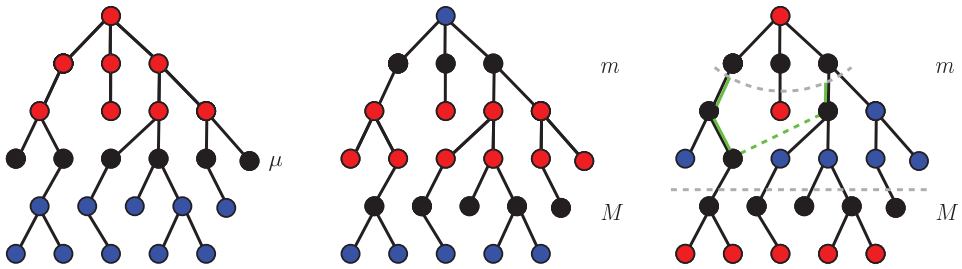

Fig. 1. The three phases of Lipton and Tarjan's algorithm (simplified illustration; nontree edges are generally omitted). From left to right: Phase 1 (simple stage; one BFS level, μ), Phase 2 (simple stage; two BFS levels, m and M), and Phase 3 (complex stage; two BFS levels plus fundamental cycle). Separator nodes are marked in black; red and blue coloring reflects component membership of the remaining nodes. The dashed gray lines cut off the nodes temporarily merged and removed during Phase 3, respectively; the nontree edge inducing the fundamental cycle (green path) is drawn dashed-green.

2.1 Algorithm Description

The classical algorithms by Lipton and Tarjan and by Djidjev both work in two stages, which we call simple and complex. The simple stage can be subdivided into two parts, Phases 1 and 2; for the sake of consistency, we will refer to the complex stage also as Phase 3. It is noteworthy that all steps of the algorithms can be implemented to run in linear time.

2.1.1 *Lipton and Tarjan's Algorithm (LT).* We now detail the single steps of the algorithm; Figure 1 gives an illustration of the different phases.

Simple Stage. Phase 1 starts by computing a BFS tree, thus grouping the nodes into levels, where the BFS root is located at level 0. Then, the middle level is defined to be the level with the smallest index μ, such that all levels with indexes at most μ encompass at least half of the graph's nodes. If this middle level contains fewer nodes than given by the separator bound, the algorithm returns that level (note that due to construction, the balance requirement is always met); otherwise, the algorithm continues with Phase 2.

In Phase 2, the levels above and below the middle level (i.e., with smaller and with greater indexes) are scanned until in each direction a level of at most $2(\sqrt{n}-D)$ nodes is found, where D is the respective distance to the middle level. These two levels, with indexes denoted by m and M, yield a separation of the remaining nodes into three parts. If the biggest of these parts and the union of the other two meet the balance bound, then the two levels are returned as a separator. Otherwise, the algorithm enters the complex stage.

Complex Stage. This stage starts with the reduction step: Temporarily all levels with indexes at least M are removed from the graph and levels 0 to m are merged to one node. Then the remaining graph is triangulated. Every nontree edge e induces a path in the BFS tree from each of e's endpoints to their "nearest" common ancestor. These paths together with e form a fundamental cycle, dividing the remaining nodes into an inner and an outer part. From all

nontree edges, the algorithm picks one arbitrarily. As long as the size of the bigger of the inner and the outer parts exceeds $2n/3$, the algorithm gradually shrinks the bigger (and enlarges the smaller) part by picking a different, neighboring nontree edge, which induces a new fundamental cycle. This shrinking step is repeated until both inner and outer parts have a size of $2n/3$ at most. The separator returned by this stage consists of the fundamental cycle and the levels m and M, and fulfills both the separator size and the component balance requirement.

2.1.2 *Djidjev's Algorithm* (*Dj*). As mentioned earlier, the algorithm by Djidjev [1982] follows the LT framework, but uses more sophisticated constructions in Phases 1 and 2, trying to find a separator consisting of one or two levels of the BFS tree, whose size has to be smaller than the one given by LT. Phase 3 is similar to that in LT.

2.1.3 *Fundamental-Cycle Separation* (*FCS*). During the experimental phase of this study, we discovered that it is very effective to give up the simple stage and apply the complex stage—that is, the algorithmic version of Lemma 1—directly to the input graph, where the reduction step is omitted. We call this the FCS algorithm. Since the height of any spanning (and hence BFS) tree is between the graph's radius r and diameter d (recall Definition 1), Lemma 1 implies that every fundamental cycle contains no more than $2d + 1$ nodes. Hence, for graphs of small diameter, the FCS approach guarantees a better bound than $\beta\sqrt{n}$.

2.2 Optimization

In practical applications of planar separator algorithms, requirements for "good separations" may vary a lot. Therefore, we provide three optimization criteria, separator size, balance, and separator ratio, which we define next and discuss how to implement them within the different stages.

Criteria. The formulation of the Planar Separator Theorem entails two natural optimization criteria, minimization of separator size, $|S|$, and maximization of component balance, $|A|/|B|$, where $|A|$ denotes the size of the smaller and $|B|$ the size of the larger component. Besides considering these individually, we also use a trade-off value, called separator ratio, introduced by Leighton and Rao [1999] and defined to be $|S|/|A|$. Consequently, as long as $|S|$ gets smaller and $|A|$ gets larger, the separator ratio becomes smaller and hence this criterion is to be minimized. The separator ratio also comes into play to break ties between separators optimized for separator size or balance.

Simple-Stage Modifications. Implementing optimization for these criteria leads to the following refinements of the algorithms. In Phase 1, picking the middle level as defined above yields good component balance in general, but no statement can be made regarding separator size. Our strategy now is simply to scan all BFS levels not violating the balance requirement and to select from amongst them the one with minimum size. A similar routine can be performed in Phase 2 (if there is a valid Phase-2 separator, the classical algorithms account

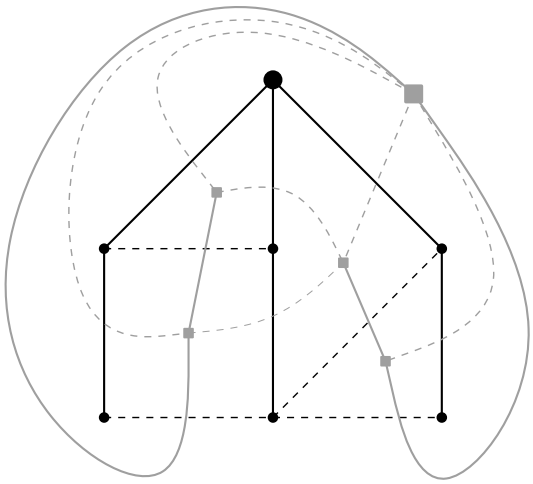

Fig. 2. Duality of spanning trees. The original graph is given in black, its dual in gray; tree edges are drawn solid, nontree edges dashed. The big gray square marks the root of the dual tree.

for optimality of neither separator size nor balance). It is fairly easy to see that these modifications do not affect linearity of the running time.

Enumerating Fundamental Cycles. The optimized version of the complex stage (and of the FCS algorithm, respectively) examines all nontree edges in the given graph, computing the sizes of the induced fundamental cycle and of the inner and outer parts, and eventually picks the best cycle according to the specified criterion. In order to do this in linear time, we need to arrange the nontree edges in such an order that examination of a subsequent cycle can rely on information computed for a preceding one; in other words, we proceed from small, inner circles to more "comprehensive" ones. Again, it can be verified easily that all construction steps require only linear time.

To order the nontree edges, we make use of the dual of the given graph through the following well-known property (see Eppstein et al. [1990] for the proof and Figure 2 for an illustration): the dual edges that correspond to the nontree edges of the original graph form a spanning tree in the dual graph. Hence, there is an immediate correspondence between cycles in the original and tree edges in the dual graph. The dual node corresponding to the outer face of the original graph is chosen to be the root of the dual spanning tree. Starting by examining the cycles in the original graph that correspond to edges incident to leaf nodes in the dual spanning tree and then continuing toward the root, the sizes of the inner and outer parts of all cycles can be computed inductively. Note that in the actual implementation, construction of the dual graph and its spanning tree is avoided. Instead, the construction of the spanning tree is simulated by performing a "breadth-first" traversal on the faces of the original graph, by use of the nontree edges. We keep track of the edges that are used to "enter" a face and then examine them in the reverse order of their discovery. The result is a correct and reasonably efficient implementation.

3. HEURISTICS

The algorithms described above are formulated at a fairly abstract level and thus bear quite some degrees of freedom, for which concrete implementing procedures have to be given: At the simple stage, BFS root and BFS order (i.e., the order in which to visit incident edges) may both be chosen arbitrarily; at the complex stage, the original, nonoptimizing formulation of the algorithms picks just any nontree edge, and the shrinking step, which has to pick an appropriate new nontree edge, is determined by the triangulation chosen.

In the following text, we propose several ways of implementing these routines, evaluated in Section 5. Regarding BFS trees, we identify some general characteristics that promote good simple and complex separators, respectively, which can be fostered through recomputation of a given BFS tree. Other than employing some standard triangulation algorithm (provided by the libraries used), we suggest a variant called triangulating BFS, which allows to simultaneously compute a triangulation and tailor a possibly more favorable tree for the complex stage. Because the tree used at the complex stage (and with the FCS algorithm) does not rely on BFS tree properties (no levels are required), any spanning tree will do; therefore, we propose an alternative way of computing such a tree. Finally, we present two postprocessing techniques to improve separations, called node expulsion and Dulmage-Mendelsohn decomposition.

3.1 BFS Variants

In this section, we describe different orders in which a graph's nodes and edges may be processed by a BFS search. Let L be the set of nodes constituting some already constructed level l. Figure 3 illustrates various BFS trees of one sample graph; note that the number of leaf nodes varies, while the tree height remains the same.

Standard Search. The nodes in L are processed in the order in which they are "stored internally": For example, these may be kept in a list (the order possibly reflecting a given embedding), which is then scanned in a linear fashion from start to end. Similar properties hold for the set of edges incident with each of L's nodes, which are often stored in clockwise or counterclockwise order. Thus, Standard Search iterates through L and for each node considers its outgoing edges; each edge being visited is included in the set of tree edges if and only if its incident node at level $l + 1$ has not been visited before.

Ordered Search. The nodes in L are sorted ascendingly or descendingly according to their degree; as mentioned previously, internal storage determines the order in which incident edges are scanned. Since typically only a few high-degree nodes of level l "cover" all nodes of level $l + 1$, descending ordering tends to generate exceptionally many leaves.

Permuted Search. This variant is similar to the ordered search; however, the nodes of level l are processed as given by a random permutation.

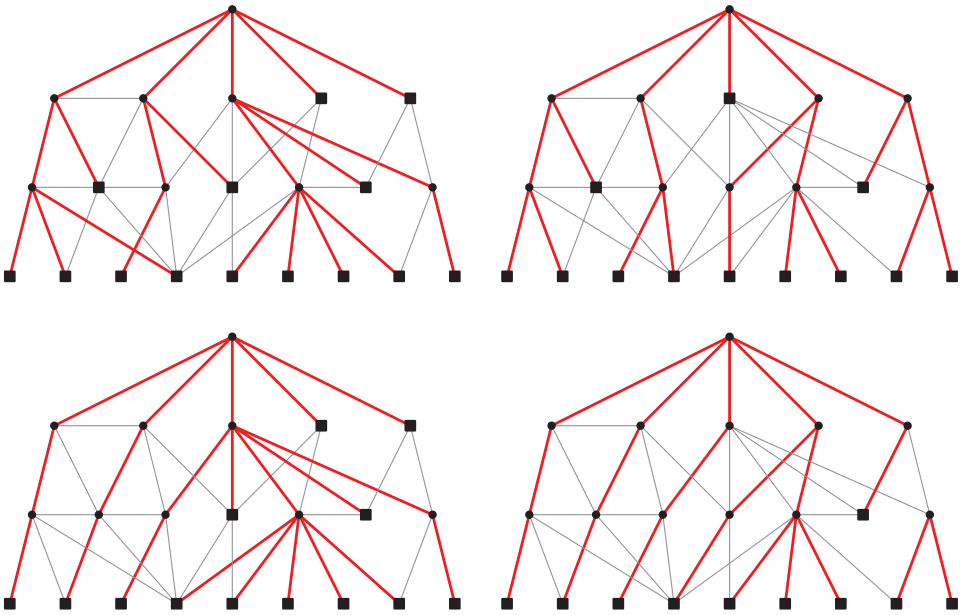

Fig. 3. BFS variants. From top left to bottom right: standard, ascendingly ordered, descendingly ordered, and balanced BFS. Bold, red lines represent tree edges; leaf nodes are marked by big squares.

Balanced Search. The intuition behind Balanced Search is to produce BFS trees that exhibit as few leaf nodes as possible; therefore, arrange the nodes in L in a circular list according to their degree. As long as there exists a node at level $l+1$ not discovered yet, iterate through the list in ascending order; if from the node being considered there is an edge to a level-$(l+1)$ node not discovered yet, declare it a tree edge; then proceed to the next node in the list.

It should be noted that all variants can be implemented to run in linear time. For Ordered Search, we can easily use bucket sort, since we know the maximum node degree of a given graph. Experiments comparing these variants are described in Section 5.3.

3.2 Tree Height Control

We want to investigate how the height of BFS trees can be affected such that they exhibit favorable properties for both the simple stage and the complex stage. Intuitively, to allow at the simple stage for a desired trade-off between small separator size and high component balance, BFS trees should have sparse middle levels (thus, balance may be finely adjusted). This implies that for the simple stage, we would like to have tall trees.

For the complex stage, recall that a separator consists of two BFS levels and a fundamental cycle. The size of the cycle is determined mainly by the height of the tree used in Phase 3, so we strive to keep this tree as shallow as possible. This is exactly the opposite goal of the simple stage. However, we are free to use a different tree at the complex stage than the one inherited from the simple

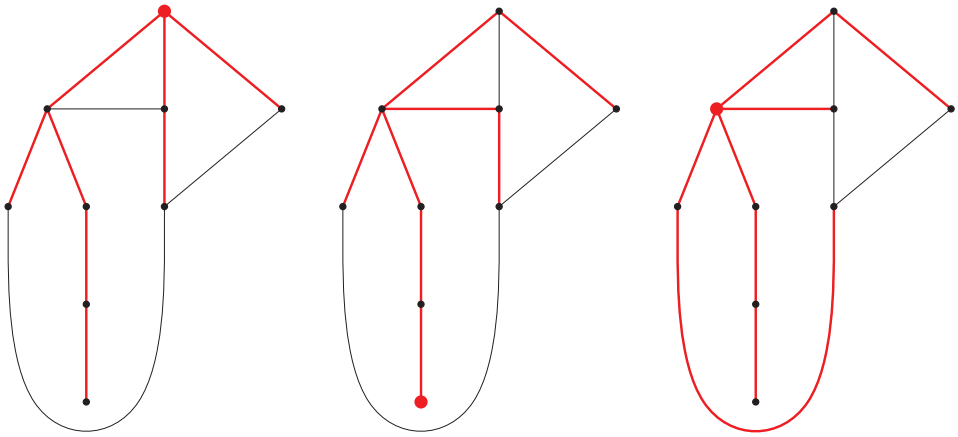

Fig. 4. Height maximization and minimization. The figures from left to right show a sample graph with a BFS tree, a BFS tree after height maximization, and a BFS tree after height minimization. BFS root and tree edges are marked in red.

stage (reducing the tree height neither invalidates the statement of the theorem nor the analysis of the algorithms).

Consequently, we introduce two procedures, called height maximization and height minimization, after which a new BFS tree is produced based on an initial one; Figure 4 illustrates these concepts.

In height maximization, we pick from the given tree a leaf at the highest level and compute a new BFS tree rooted at this node. This process may be iterated for a constant number of times. It is clear that, by construction, each iteration does not reduce the tree height.

Height minimization proceeds in a similar way but chooses as a new root node a centroid of the tree, that is, a node whose maximum distance to any other node in the tree is minimum (each tree contains either one or two centroids, which can be determined in linear time). By definition, this procedure delivers a new tree of equal or smaller height. Results from an experimental evaluation of these heuristics are given in Section 5.3.

3.3 Triangulating BFS

As described in Section 2.1.1, triangulation takes place in the graph obtained from the reduction step (merging and removal of nodes). On the other hand, as just seen, the BFS tree induced by this reduction can be replaced by a new one, which implies that tree edges may at the same time be triangulation edges.

These insights are exploited by the subsequent process. Since for the complex stage shallow trees are desirable, the more adjacent nodes a BFS reaches from a single node, the smaller the tree height potentially becomes. On the reduced graph, we now start a new BFS (with arbitrary root): From each node v whose incident edges are being explored, additionally introduce a triangulation edge to each node w that is not connected to v but can be linked through an edge without causing any crossings; if w has not been visited yet, make $\{v, w\}$ a

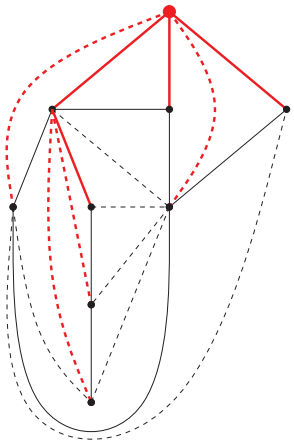

Fig. 5. Triangulating BFS applied to the graph from Figure 4. BFS root and tree edges are marked in red, triangulation edges are dashed.

tree edge. Continue this search until the graph is triangulated (which may well be done after the tree has been constructed). Figure 5 shows an illustration of this method.

3.4 Star Trees

The reason for which the algorithm by Lipton and Tarjan uses BFS trees instead of arbitrary ones is that each BFS level has the property of separating the graph into two parts. This feature is exploited during the simple stage; however, it is not required for the complex stage, which suggests trying also different trees.

An alternative idea is to rely on a bottom-up procedure that "greedily covers" the graph used for the complex stage with isolated stars (i.e., trees of height at most 1 not connected to one another) and grows these together to a maximal tree; roughly, at each step, a node connected through nontree edges to many other partial trees is chosen and these nontree edges are made tree edges. However, preliminary experiments showed that the trees thus arising cannot compete with our BFS trees, so we eventually discarded this approach.

3.5 Postprocessing

In the following text, we provide two postprocessing steps that can be used to improve the quality of separations in terms of separator size and/or component balance.

Node Expulsion. This technique consists in moving nodes of a simple separator that do not separate two nodes from different components to the smaller component, which improves both separator size and component balance (see Figure 6).

Dulmage-Mendelsohn Optimization. The idea of this technique is to shift a subset of the separator nodes to their adjacent nodes in the larger component

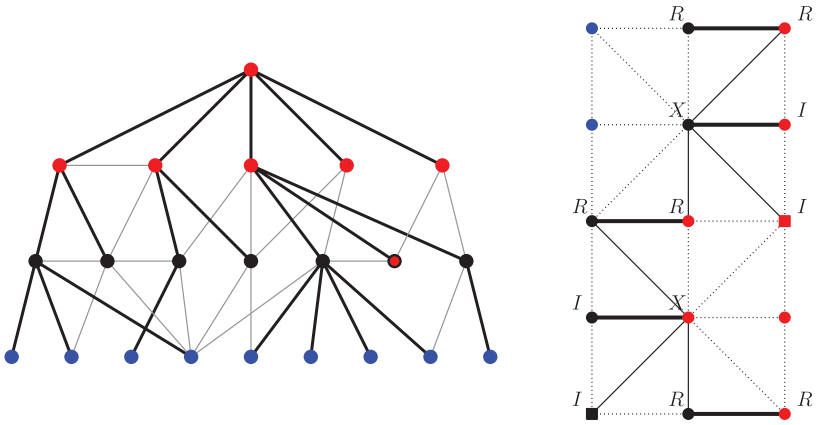

Fig. 6. Postprocessing techniques. Black nodes are separator nodes, while blue (and red, respectively) nodes are the nodes of the blue (and red, respectively) component. Node expulsion (left): The bold edges form a BFS tree of the given graph. The separator node connected to the red, but not to the blue component (red disk with black border) can be moved to the red component, which improves both separator size and component balance.

Dulmage-Mendelsohn optimization (right): Solid lines indicate edges of the bipartite graph induced by the black and red nodes, bold edges show a maximum matching in this graph, where squares highlight unmatched nodes; the labels denote membership of both S and Adj to the respective node sets internal (I), external (X), or residual (R). Shifting the separator nodes in I to the red component reduces the separator size by 1 and improves component balance from 2:7 to 4:6 (blue/red nodes), while shifting the separator nodes in I and R to the red component yields equal reduction in separator size, but a 7:3 balance.

[Ashcraft and Liu 1996]. In other words, detect a subset of the separator, $\emptyset \neq S' \subset S$, such that the subset $B' \subset B$, consisting of nodes that are adjacent to S' and belong to the larger component B, is smaller than S'. Then, the separator is modified by removing the nodes in S' and adding the nodes in B'. The size of the new separator is smaller than the original one, and the balance may be improved as well. This is done by computing a maximum-cardinality matching on the bipartite graph induced by the separator S and its adjacent nodes, $Adj_B(S)$, that belong to the larger component. Either set of nodes constituting this bipartite graph is divided into three groups: a node in S is called internal (I) if it is reachable via alternating paths from an unmatched node in S; external (E) if it is reachable via alternating paths from an unmatched node in $Adj_B(S)$; and residual (R) otherwise—a symmetric definition holds for the nodes in $Adj_B(S)$ for an illustration, (see Figure 6). Shifting either S_I or $S_I \cup S_R$ to their adjacent nodes in the bipartite graph gives maximum reduction in separator size.

4. GRAPHS

The following list describes all graph classes used for our experiments, mostly randomly generated but also from real world, where similar classes are gathered in one paragraph. Examples of a few graph classes are depicted in Figure 7. To enhance the significance of our investigation, we employ a set containing one

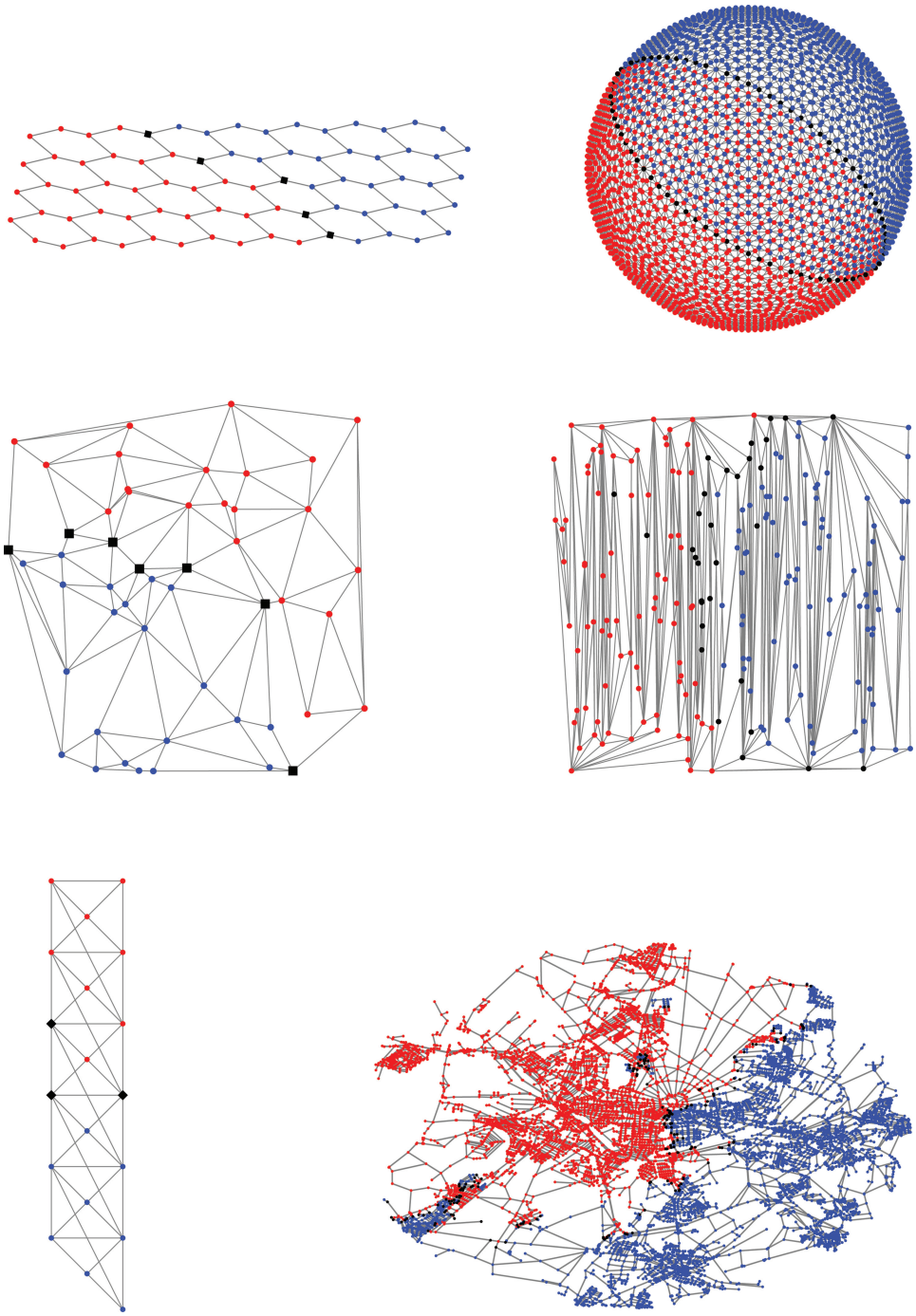

Fig. 7. Sample graphs with separators. From top left to bottom right: `hex`, `t-sphere`, `del`, `leda`, `diam`, and `ka`. Separator nodes are drawn in black, nodes attributed to one component in red, and nodes attributed to the other component in blue.

Table I. Graph Parameters: Number n and m of Nodes and Edges, Diameter and Radius for Both the Original and Triangulated Graphs

Graph	n	m	Diameter orig	Diameter triang	Radius orig	Radius triang	Remark
square	10,000	19,800	198	67	100	50	
rect	10,000	19,480	518	20	260	10	20×500 raster
hex	9,994	14,733	513	22	257	11	20×237 raster
globe	10,002	20,100	101	101	76	67	100×100 circles
t-sphere	10,242	30,720	96	96	80	80	5 iterations
diam	10,000	29,994	3,333	3,333	1,667	1,667	
del	10,000	25,000	56	45	46	34	
del-max	10,000	29,971	52	48	43	39	
leda	9,990	25,000	18	14	11	8	
leda-max	10,000	29,975	16	16	9	8	
c-square	10,087	19,904	219	71	110	36	5 connecting nodes
c-del-max	10,005	29,972	62	55	33	28	5 connecting nodes
c-leda-max	10,005	29,984	18	15	9	8	5 connecting nodes
ka	10,298	14,175	129	28	69	18	

graph of each class with roughly the same size; a detailed synopsis can be found in Table I.

Grid Graphs. An obvious choice of regularly structured planar graphs is to use grids of different shapes:

—square and rect graphs are $(x \times x)$- and $(x \times y)$-rasters of nodes, respectively, where adjacent nodes of the same row or column are connected by an edge. Straightforward calculation shows that a square with n nodes has a minimum simple separator of approximately $\sqrt{2n/3} \approx 0.82\sqrt{n}$ nodes.

—hex graphs can be seen as a number of hexagons "glued together" in a honeycomblike fashion. As can easily be verified by Figure 7, a grid of $x \times y$ honeycombs contains $2(x+1)y + 2x$ nodes. Clearly, hex graphs are the sparsest of all grid graphs.

Sphere Graphs. In Djidjev [1982], a class of graphs with a lower bound of $1.56\sqrt{n}$ on separator size is introduced, which is obtained through approximation of the unit sphere. With our experiments, we consider two similar classes of graphs, whose generation is a little more straightforward.

—globe graphs are induced by equally distributed circles of longitude and latitude of a unit sphere, where nodes are induced by edge crossings.

—t-sphere graphs approximate the unit sphere by triangles (see Bourke [1992]). The iterative generation process starts with an icosahedron (consisting of 20 equilateral triangles with all nodes on the sphere); at each step, each triangle is split into four smaller, identical ones by placing a new node in the middle of each edge and interconnecting these through edges.

Big-Diameter Graphs. Under the name of diam, we provide graphs exhibiting rather big diameters: Given an integer d, a maximal planar graph with

$3d + 1$ nodes and of diameter d is generated. By construction, `diam` graphs have separators of size 3.

Random Graphs. Random planar graphs come in two flavors, according to the triangulation used to generate them:

—`del` and `del-max` graphs employ a Delaunay triangulation, where quite a regular distribution of node degrees is achieved.
—triangulation of `leda` and `leda-max` is computed by the LEDA library (see Näher and Mehlhorn [1999]): roughly, a sweep-line algorithm scans the nodes and inserts edges as needed, causing many (almost-)vertical edges.

Construction of these graphs is done by placing at random a given number of nodes in the plane and triangulating the convex hull, which immediately yields the respective `-max` variant. The general variants, `del` and `leda`, are obtained by deleting from the maximal graph a desired number of edges.

Small-Separator Graphs. To construct graphs with small separators yielding perfect balance, we connect two copies of a graph through a few additional nodes; the challenge for our algorithms then is to find this separator. These graphs are indicated with a `c-` prefix (for connected); for our study, we employ `c-square`, `c-del-max`, and `c-leda-max`.

Real-World Graphs. Graphs representing road networks typically have the property of being almost planar (edge crossings are mostly caused by bridges/underpasses, given the geographic embedding). Such edge crossings can be removed by simply placing a node on it, without altering the graph too much. For our experiments, we use a graph extracted from the German road network,[1] denoted by `ka`, that represents the city of Karlsruhe.

5. EXPERIMENTS

Our empirical study involving the aforementioned base algorithms, heuristic methods, and graphs was undertaken in an inductive fashion, where first a broader range of combinations was investigated, followed by more specific parameter settings under consideration of preceding results. The presentation of our findings is thus divided into five parts:

—In order to obtain a comprehensive overview of the performance of each algorithm optimized for the different criteria and applied to each graph class, we started by running a complete series of these combinations, investigating separator size, component balance, separator ratio, and terminating phase.
—To get a finer picture of algorithmic performance, we studied a series of graphs increasing in size, taking into account also running time.
—We next focused on the heuristics for BFS search and triangulation described in Section 3.

[1]The data was provided courtesy of PTV AG, Karlsruhe.

—We evaluated the postprocessing techniques, introduced in Section 3.5, with respect to reduction of separator size for a few selected graphs.
—The final part summarizes our experimental outcomes, and deduces from the preceding observations some general recommendations for the choice of algorithms and parameters when one wishes to separate a given graph for a desired criterion.

We implemented the algorithms in C++, using the LEDA (version 5.0.1) and Boost (version 1.33.1) libraries. Our code was compiled with the GCC (version 3.4), and executed on different Intel Xeon and Opteron machines, running a Linux kernel.

5.1 Algorithmic Comparison

The first series of experiments focuses on our algorithms LT, Dj, and FCS, optimized for the different criteria (separator, balance, and ratio), as described in Section 2.1 and applied to the graphs listed in Table I. Each node is once picked as the root for BFS search. Our experimental results display separator size, component balance, and separator ratio with each algorithm, both unoptimized and optimized for the respective criterion, and are presented in the form of boxplots: For each combination of algorithm and graph, the corresponding box represents the middle 50% of values obtained over all root nodes, each whisker spans a range of 1.5 times the height of the box (outliers are not shown), and average values are marked by a cross.

Separator Size. The plots of Figure 8 denote relative separator size, that is, the absolute number of nodes in a separator divided by the square root of the number of nodes in the graph (which corresponds to β from Section 2.1). The graphs are distinguished along the x-axis; for each graph, the algorithms LT, Dj, and FCS are colored (from left to right) red, blue, and green, respectively. In general, it can be observed that for most graphs tested, average separators are significantly smaller than the respective upper bound, especially for Dj. For LT, the hardest instances are the `(c-)del(-max)` graphs, for which the bound is actually reached, and even average separator sizes are beyond $3\sqrt{n}$.

Overall, FCS performs best: For almost all graphs, both maximum and average separators are smaller than those achieved by the other two algorithms (exceptions hereto are, for the unoptimized case, the sphere graphs and—to some negligible extent—the `diam` graph). This result holds not only for instances with rather small diameters (according to Table I, triangulated `del` exhibits a diameter of 45, so the theoretic bound of $2 \cdot 45 + 1 = 91$ is considerably smaller than Djidjev's bound of $\sqrt{6 \cdot 10{,}000} = 245$), but also for `diam`, which features a much larger diameter.

Concerning the unoptimized algorithms, great reduction in separator size (up to a factor of around 5) can be achieved for `del(-max)` and `leda(-max)` when applying Dj instead of LT, while similar savings are obtained for `ka` when switching from Dj to FCS. With optimization employed, the differences between the algorithms are slightly less pronounced: For the grid and `ka` graphs,

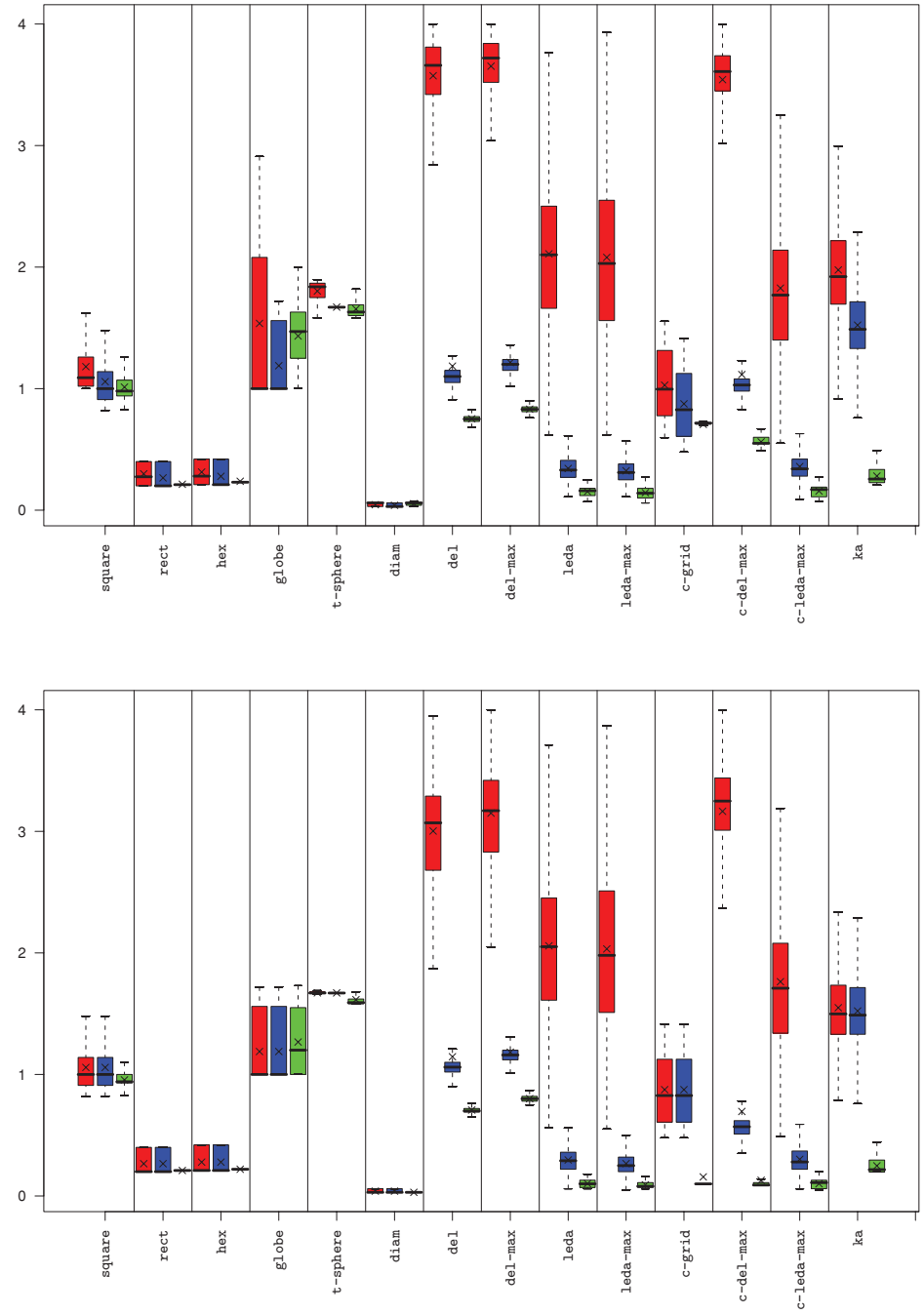

Fig. 8. Relative separator size (absolute number of nodes divided by the square root of the number of nodes in the graph) for the 10,000-node graphs with the different algorithms, both unoptimized (top) and optimized for separator size (bottom). The graphs are distinguished along the x-axis; for each graph, the algorithms LT, Dj, and FCS are colored (from left to right) red, blue, and green, respectively.

performance of LT and Dj is almost identical; for the random graphs, Dj and especially FCS work considerably better. Regarding improvement with optimized vs. unoptimized algorithms, LT often yields smaller mean values (e.g., for the grid, `del`, and `ka` graphs) or smaller boxes (e.g., for `globe`); separator reductions obtained for Dj and FCS, in contrast, are generally marginal, but for FCS applied to `c-square`, an improvement from roughly 0.7 to 0.2 is achieved.

Component Balance. Component balance (size of the smaller component divided by the size of the larger one) is shown in Figure 9. Unlike with separator size, there is a large difference between the unoptimized and optimized variants: On average, LT gives good or excellent balance even when no optimization is applied, except for the `(c-)leda(-max)` and `c-del-max` graphs. However, this is not true for Dj and especially FCS applied to many instances, where for the unoptimized variant poor balance is often achieved, while the optimized algorithms give mean balance of at least 0.9 for all graphs.

This behavior can easily be explained through the fact that for many graphs, LT succeeds in determining a Phase-1 separator (consisting of one BFS level dividing the graph into two components of similar size)—except for `(c-)leda(-max)` and `c-del-max`, which mostly demand for Phase 3 (see Figure 11). On the other hand, the initial cycle separator computed by FCS does not necessarily guarantee high component balance. Allowing the algorithm to level out imbalance by appropriately shrinking the bigger component, however, may produce even better separators than achieved by LT; in particular, with graphs exhibiting small separators (e.g., the `c-`graphs) balance can be better fine-tuned by FCS.

Separator Ratio. The separator ratio, reflecting the ratio between separator size and component balance, is shown in Figure 10 and is naturally correlated to the previous plots, where the overall picture strongly follows that for separator size. Altogether, FCS is a favorable choice; only with single graphs, such as `globe`, comparatively large separator size adds to rather poor balance, so that FCS does not perform best even when optimization is applied. A similar observation holds for Dj, so that for a few graphs, the unoptimized LT slightly outperforms Dj. Interestingly, some worsening in both maximum and mean ratio can be observed for quite some graphs when LT is optimized: This reflects the tradeoff between finding a (Phase-1) separator that is both small and induces high balance.

Terminating Phase. Figure 11 finally shows for the classic algorithms the share of each phase in all root nodes for which that phase suffices to find a valid separator with LT and Dj, respectively. The following distinction can be made: regular-structured and fairly sparse graphs (especially the grid, sphere, `diam`, `c-square`, `c-globe`, and `ka` graphs) generally require only Phase 1; in contrast, the `(c-)leda(-max)` graphs cannot be separated through a simple separator, but require phase 3 with both LT and Dj. A border case is constituted by `del` and optimization for separator size, for which LT always terminates after Phase 1, while Dj applies mostly Phase 3. This can be explained due to the fact that the average BFS level contains roughly $300 = 3\sqrt{10{,}000}$ nodes, which exceeds the

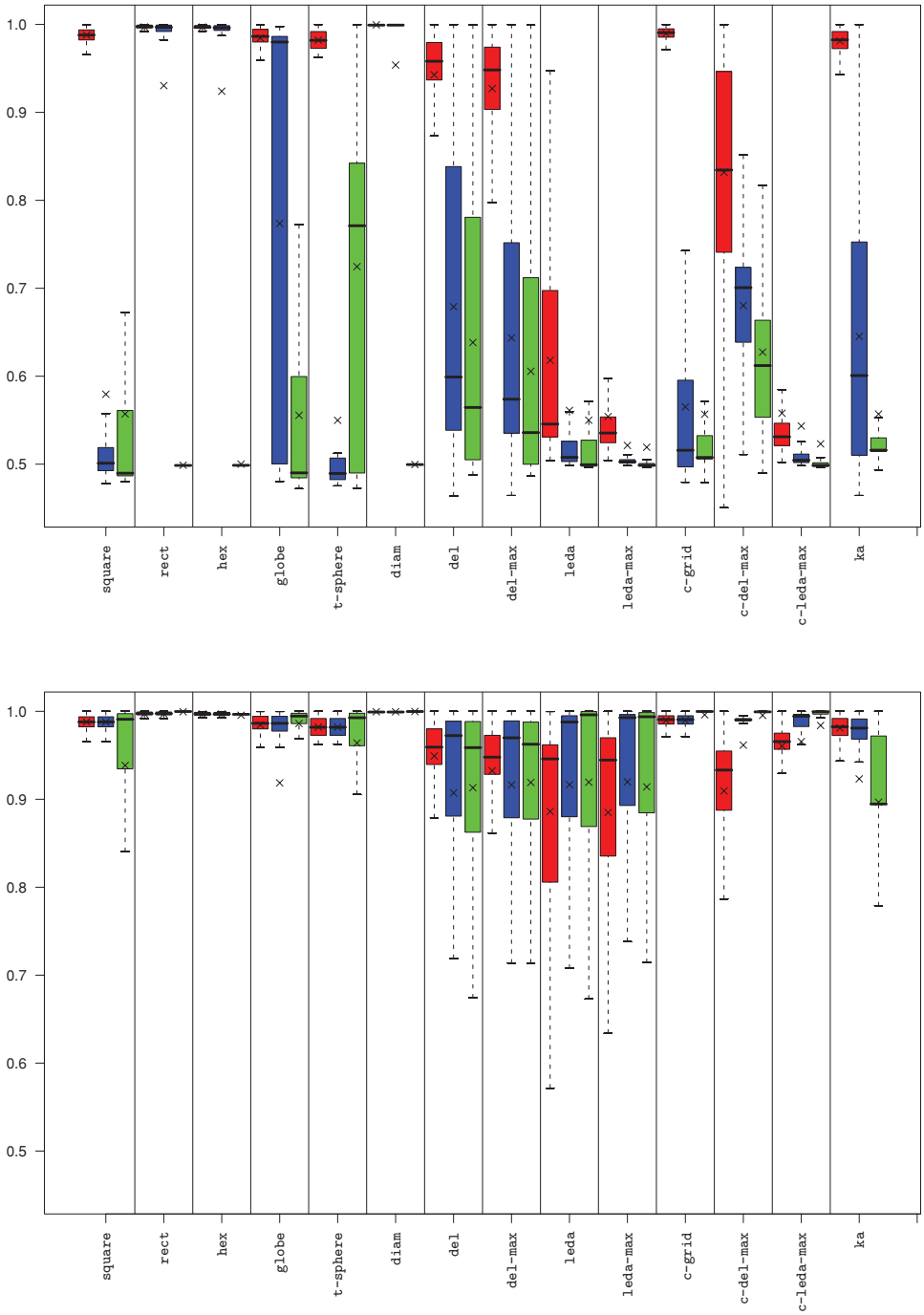

Fig. 9. Component balance for the 10,000-node graphs with the different algorithms, both unoptimized (top) and optimized for balance (bottom). The graphs are distinguished along the x-axis; for each graph, the algorithms LT, Dj, and FCS are colored (from left to right) red, blue, and green, respectively.

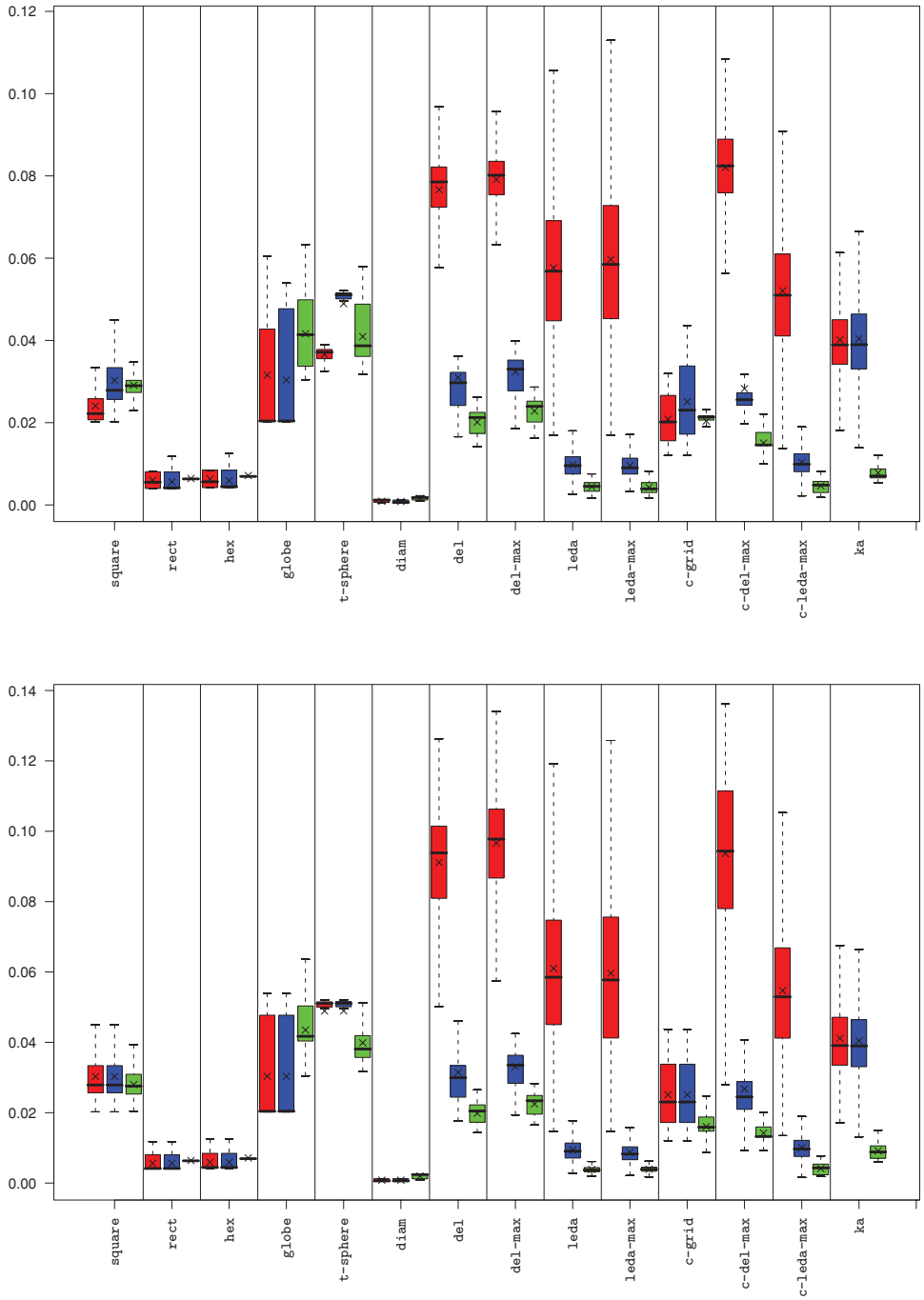

Fig. 10. Separator ratio for the 10,000-node graphs with the different algorithms, both unoptimized (top) and optimized for ratio (bottom). The graphs are distinguished along the x-axis; for each graph, the algorithms LT, Dj, and FCS are colored (from left to right) red, blue, and green, respectively.

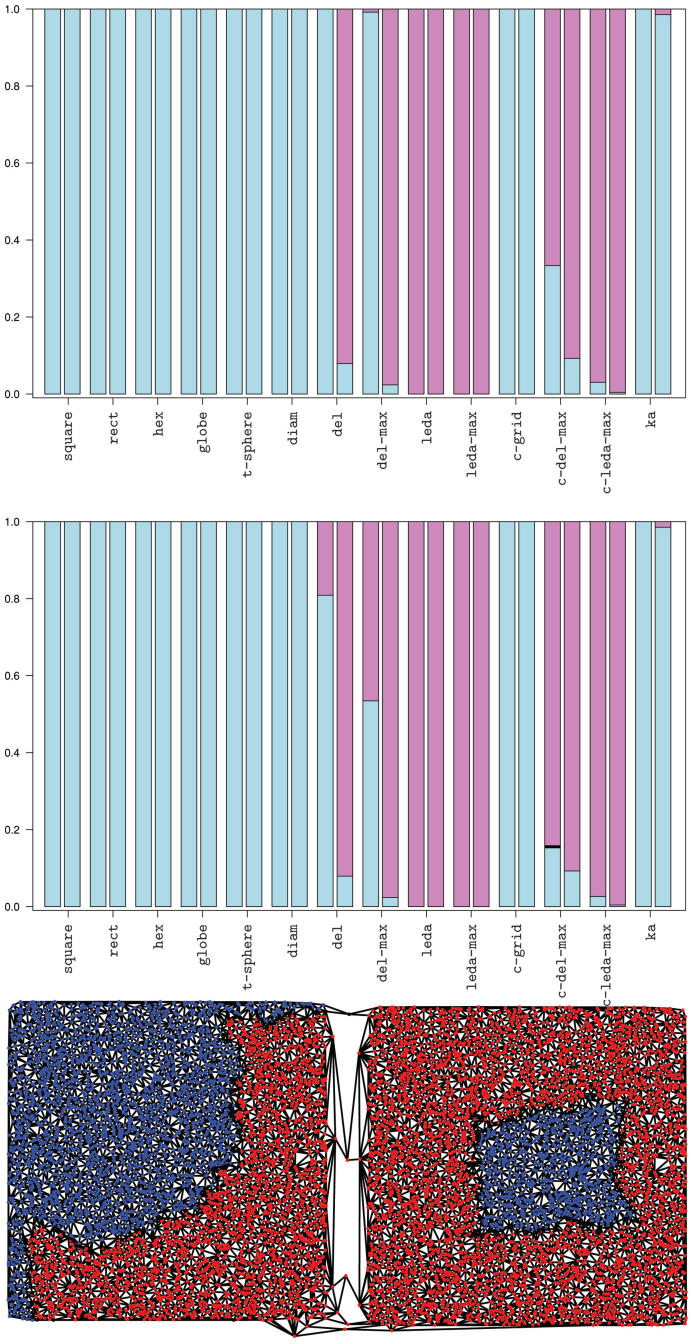

Fig. 11. Top and middle: Shares of terminating phases for the 10,000-node graphs with LT (left bar of each group) and Dj (right bar) optimized for separator size (top) and component balance (middle). The graphs are distinguished along the x-axis; light-blue, black, and pink segments represent Phases 1, 2, and 3, respectively. Bottom: Snapshot of the `c-del-max` graph with a Phase-2 separator (black nodes, dividing the red from the blue component).

bound accepted by Dj. When optimizing for balance, however, LT also has to enter Phase 3 for around 20% of root nodes.

It is extremely surprising to discover that there are only very few cases in which LT is terminated in Phase 2: For `c-del-max`, $< 1\%$ of root nodes induce Phase-2 separators when optimization for balance is applied; a snapshot of an example is shown in Figure 11.

5.2 Graph Series

To study the behavior of our algorithms with a series of graphs increasing in size, we chose the `del` class, considering 20 instances with a number of nodes between 50 and 1,000 and an edge density of 2.5 times the number of nodes. The advantage of this choice is that random instances of arbitrary size can be generated that are not totally regular in structure; furthermore, due to the previously mentioned outcome, these graphs seem to constitute rather hard instances for LT, while leading to a varying terminating phase. Again, we take into account all three algorithms, however, without any optimization applied, and iterate over all BFS root nodes, where we now also iterate over all possible nontree edges. Our results are reported in Figures 12 through 14, which depict terminating phase, separator size, running time, as well as the impact of BFS root node and nontree edge selection on separator size and component balance. In the following text, we discuss in detail the reported results.

Separator Size. The upper diagram of Figure 12 shows the relative separator size of the different algorithms, along with the terminating phase in the lower diagram. For increasing numbers of nodes, we observe a growth in separator size from $2\sqrt{n}$ to just under $3\sqrt{n}$ for LT, while for FCS, separator size appears fairly stable (after a slight decline with the first few instances). The strong decline with Dj can be explained by the fact that with increasing frequency valid separators are found only in Phase 3: Obviously, the levels obtained from a BFS become larger, so that simple separators still meet the bound for LT, but fail to satisfy the bound for Dj.

Running Time. Running time is shown in Figure 13: The upper diagram depicts the total running time of each algorithm over all root nodes and all nontree edges. Although linearity of the running time can be clearly confirmed for all algorithms, the actual factors determining the time spent by a processor vary enormously: LT, which we have shown to always terminate in Phase 1, consumes for each graph $<$1ms; Dj, increasingly involving Phase 3, takes up to 30ms; and for FCS, a maximum running time of just over 60ms is obtained.

To get a better understanding of the time spent for FCS (and the complex stage, respectively), the average running time with FCS is broken down to different subroutines, as shown in the lower diagram of Figure 13. These subroutines include initialization, embedding, and triangulation of the input graph, BFS tree computation, determination of the order of nontree edges (see Section 2.2), initial computation and possibly enhancement of a cycle separator, and residual tasks (including control structures, method calls, etc.). It becomes

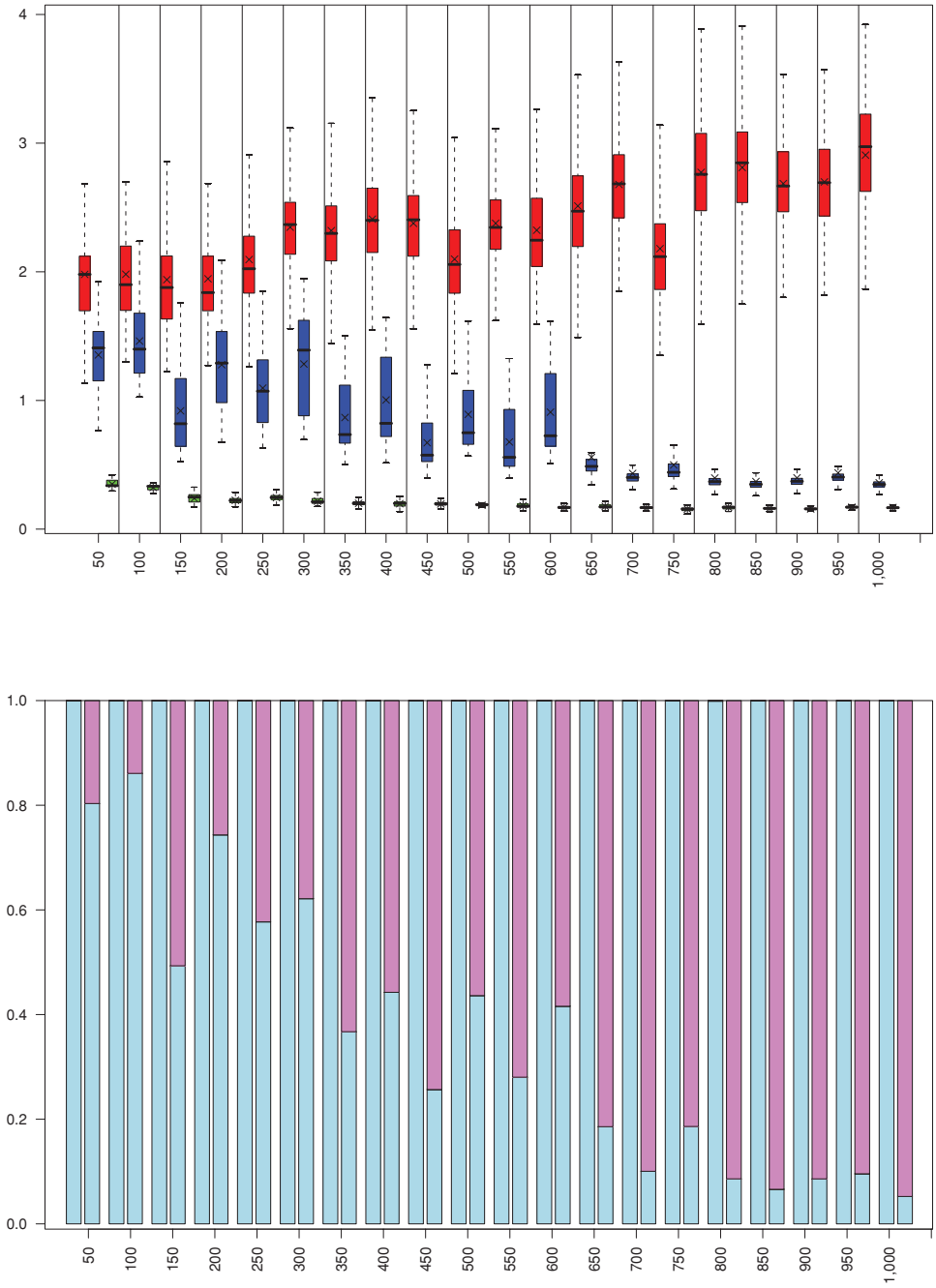

Fig. 12. Separator size (top) and terminating phase (bottom) for a series of 20 `del` graphs. The labels on the x-axis indicate the numbers of nodes in the graphs; the arrangement and colors match the ones used for Figures 8 and 11.

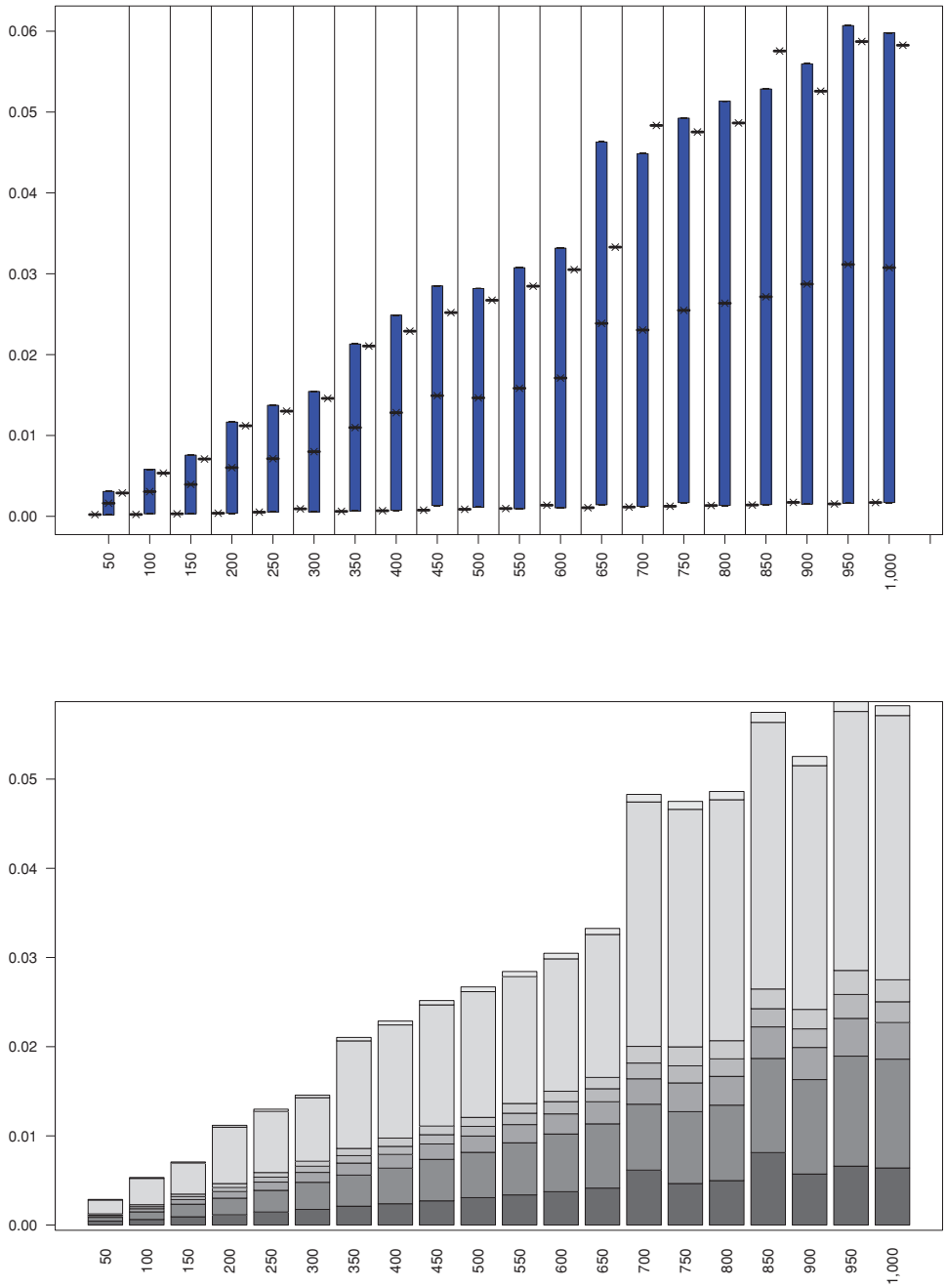

Fig. 13. Total running time (top) and running time for FCS (bottom) in seconds for a series of 20 `del` graphs. The labels on the x-axis indicate the numbers of nodes in the graphs; the arrangement matches the one previously used. In the lower diagram, the different segments reflect (from dark- to light-gray colors and bottom to top, respectively) times spent on initializing, embedding, and triangulating the graph, computing a BFS tree, determining the order of nontree edges, constructing/improving fundamental cycles, as well as residual tasks.

clear that iterating over all nontree edges (including determination of the order of nontree edges) comprises well over 50% of the running time. However, note that such an iteration is indispensable when an optimization criterion is used.

Influence of Choices. The upper diagram in Figure 14 shows the influence of BFS root and nontree edge selection, respectively, on separator size: For each graph and application of FCS, the range of mean relative separator size (marked by a circle in the diagram) is determined by computing for each BFS root the average relative separator size over all possible nontree edges and taking the difference between the maximum and the minimum of these average values; consequently, high numbers of this measure correspond to a great impact of BFS root selection. A similar measure (marked by triangles) denotes the range, over all nontree edges, of average relative separator sizes over all BFS nodes, which reflects the influence of nontree edge selection. Analogously, the lower diagram shows these values for balance instead of separator size. The diagrams suggest that the variation of separator size depends to a similar extent on BFS root and on nontree edge selection, whereas the more decisive factor for balance seems to be nontree edge selection.

5.3 Heuristics

The preceding experiments revealed that the FCS algorithm generally outperforms the others, especially in terms of separator size, but its running time is slower than that of LT by significantly more than one order of magnitude. The LT algorithm, on the other hand, yields quite good balance (partly owing to the fact that most graphs settle for Phase 1), but considerably worse results in terms of separator size. In addition, the separator size heavily depends on BFS root selection; observe the relatively large boxes in Figure 8.

Therefore, we evaluate in the following whether variation of the BFS coupled with height maximization, as introduced in Section 3, might be suited to enhance performance of LT. Moreover, we present a very small examination of triangulating BFS. Due to the overwhelming number of possible combinations, we report our results for one graph class using a few parameters and respecting only the separator size. Similar results hold for the other graph classes considered.

Figure 15 shows relative separator size with the `del` graph and different combinations of BFS order and number of iterations for height maximization: All combinations involving height maximization reduce the mean separator size by almost 10%, and also the maximum separator size is reduced. On the other hand, BFS order does not seem to matter, and also 10 steps of iteration do not yield any improvement over 2.

Experiments with triangulating BFS in contrast to individual triangulation followed by (standard) BFS showed that a reduction in relative separator size from 0.75 to 0.68 is attainable for `del`, using a random sample of five roots for triangulating BFS and 15 iterations for height minimization.

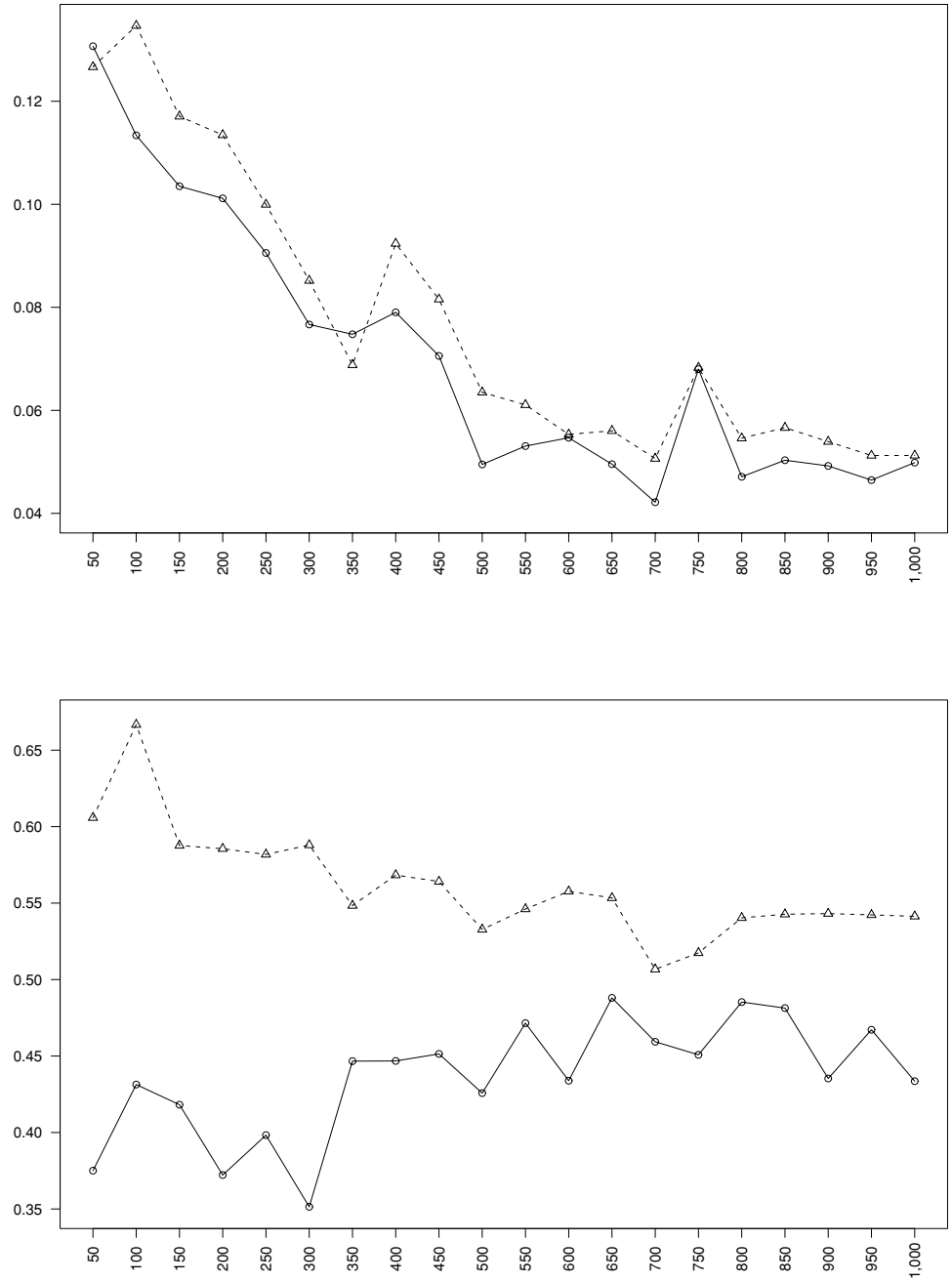

Fig. 14. Influence of the choice of BFS root and nontree edge on separator size (top) and balance (bottom) for a series of 20 `del` graphs. The labels on the x-axis indicate the numbers of nodes in the graphs; the y-axis denotes the average range of relative separator size and balance, depending on the choice of BFS root (circles, solid lines) and nontree edge (triangles, dashed lines), respectively.

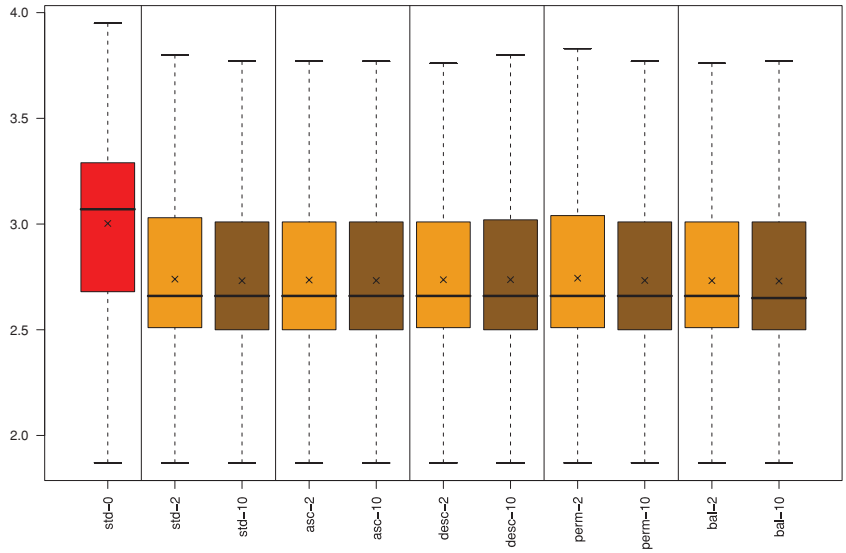

Fig. 15. BFS variation and height maximization with the 10,000-node `del` graph. The y-axis denotes relative separator size, the labels on the x-axis give BFS order and number of iterations for height maximization. The colors correspond to the number of iterations.

Table II. Reduction in Relative Separator Size Through the Node Expulsion Postprocessing for Different 10,000-Node Graphs

Graph	Before	After
`del`	3.00	2.35
`del-max`	3.18	2.61
`c-square`	0.88	0.88
`ka`	1.57	1.21

5.4 Postprocessing

Preliminary results (see Holzer et al. [2005]) suggested that all possible combinations of our two postprocessing techniques give similar reduction in separator size. Combination of the node expulsion followed by the Dulmage-Mendelsohn optimization is slightly superior to mere node expulsion, but more expensive by a few orders of magnitude. In fact, our implementation of node expulsion takes less than an additional 3% of the time used by LT to compute a primary separator for `del`, whereas postprocessing with the Dulmage-Mendelsohn optimization requires in addition (the enormous amount of) 20 times the initial LT running time.

Table II lists, for a few graphs settling for Phase 1 with LT, the average relative separator size (over all root nodes) before and after postprocessing. Effectiveness of the preprocessing naturally depends on the graphs: For `c-square`, no significant improvement can be obtained but for `del`, savings of well over 20% are attainable.

5.5 Recommendation

The above experimental outcomes suggest the following recommendations for the choice of separator algorithms, depending on the optimization criterion used. Note that all procedures suggested require only linear time.

—When optimizing for separator size and running time is not a critical issue, we suggest to use FCS with triangulating BFS and height minimization with a few iterations.

—When optimizing for separator size and running time does matter, it might be an option to employ LT, especially for graphs that promise to yield simple separators with high probability. Moreover, the use of height maximization (with only a few iterations) and node expulsion postprocessing are strongly recommended.

—When optimizing for component balance, an effective and fast algorithm for many graphs arising in practice is LT. For graphs that require careful fine-tuning of component balance, FCS is suggested.

6. DISCUSSION

In this study, we have thoroughly investigated the Planar Separator Theorem by Lipton and Tarjan (as well as—to some minor extent—the further development by Djidjev) with respect to several degrees of freedom. For each of these parameters, we provided concrete variants for implementation and also suggested a few simple methods to heuristically improve separator quality. We further proposed a new, straightforward separator algorithm relying on only part of the aforementioned algorithms. Experiments on a variety of graphs have shown, among other things, the following:

—For many of the graph classes tested, the classical algorithms often require only Phase 1 to yield valid separators, where sizes are typically much smaller than the theoretic bounds suggest. Moreover, Phase 2 is only seldom used.

—On the other hand, for quite a few graphs there is a high potential for optimization, which is confirmed by the large range in separator size with different BFS roots. Using the height maximization heuristic (possibly coupled with choosing a small sample of initial roots for BFS) can mitigate this effect considerably.

—For almost all graphs, smaller separators are reached through FCS. Selection of a nontree edge at the complex stage may greatly influence both separator size and component balance, where our implementation allows to select in linear time an edge that optimizes a given criterion.

—Merging the triangulating step, needed at the complex stage in order to shrink fundamental-cycle separators, with recomputing a BFS tree (and additionally applying height minimization) can further improve separator size.

Based on these findings, we see the following issues for further investigation:

—Another step of the algorithms whose implementation is not specified concerns the embedding of the input graph, which in turn may influence the

set of triangulations considered. In our experiments, we naturally chose the embedding induced by the graph's coordinates; however, it is not quite clear whether a different embedding would exhibit "more advantageous" triangulations. Enumerating all possible embeddings and triangulations is not an option, so one would have to find some formulation for the more promising constellations to narrow down the set of options.

—As for theoretical complexity, it has been known for quite some time that finding minimum-size separators bisecting arbitrary graphs is $\mathcal{NP}$-hard; however, a similar result for planar graphs was shown only recently, in Fukuyama [2006]. One practical approach to determining optimum separators, at least for small graphs, could be a simple enumeration algorithm with some branch-and-bound strategy.

ACKNOWLEDGMENTS

The authors would like to thank their students Imen Borgi, Jürgen Graf, Jens Müller, and Tirdad Rahmani for their great assistance and manyfold ideas with the implementation and experimentation.

REFERENCES

ALBER, J., FERNAU, H., AND NIEDERMEIER, R. 2003. Graph separators: A parameterized view. *J. Comput. Syst. Sci. 67*, 4, 808–832.

ALEKSANDROV, L., DJIDJEV, H., GUO, H., AND MAHESHWARI, A. 2006. Partitioning planar graphs with costs and weights. *ACM J. Exp. Algorithms 11*.

ALON, N., SEYMOUR, P., AND THOMAS, R. 1994. Planar separators. *J. Discrete Math. 7*, 2, 184–193.

ASHCRAFT, C. AND LIU, J. W. H. 1996. Applications of the Dulmage-Mendelsohn decomposition and network flow to graph bisection improvement. Tech. rep. CS-96-05, Department of Computer Science, York University, North York, Ontario, Canada.
`http://www.cs.yorku.ca/techreports/1996/CS-96-05.html`.

BOURKE, P. 1992. Sphere generation.
`http://astronomy.swin.edu.au/pbourke/modelling/sphere/`.

DJIDJEV, H. N. 1982. On the problem of partitioning planar graphs. *J. Algebraic Discrete Methods 3*, 2, 229–240.

DJIDJEV, H. N. AND VENKATESAN, S. M. 1997. Reduced constants for simple cycle graph separation. *Acta Informatica 34*, 231–243.

EPPSTEIN, D., ITALIANO, G. F., TAMASSIA, R., TARJAN, R. E., WESTBROOK, J., AND YUNG, M. 1990. Maintenance of a minimum spanning forest in a dynamic planar graph. In *Proceedings of the 1st Annual Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, 1–11.

FREDERICKSON, G. N. 1987. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput. 16*, 6, 1004–1022.

FUKUYAMA, J. 2006. Np-completeness of the planar separator problems. *J. Graph Algorithms Appl. 10*, 2, 317–328.

GOODRICH, M. T. 1995. Planar separators and parallel polygon triangulation. *J. Comput. Syst. Sci. 51*, 3, 374–389.

HENZINGER, M. R., KLEIN, P., RAO, S., AND SUBRAMANIAN, S. 1997. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci. 55*, 3–23.

HOLZER, M., PRASINOS, G., SCHULZ, F., WAGNER, D., AND ZAROLIAGIS, C. 2005. Engineering planar separator algorithms. In *Proceedings of the 13th Annual European Symposium on Algorithms*. Springer, Berlin, 628–639.

KOZEN, D. 1992. *The Design and Analysis of Algorithms*. Springer, Berlin.

LEIGHTON, T. AND RAO, S. 1999. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM 46*, 6, 787–832.

LIPTON, R. AND TARJAN, R. E. 1980. Applications of a planar separator theorem. *J. Comput. 9*, 615–627.

LIPTON, R. J. AND TARJAN, R. E. 1979. A separator theorem for planar graphs. *J. Appl. Math. 36*, 2, 177–189.

MEHLHORN, K. 1984. *Data Structures and Algorithms 1, 2, and 3*. Springer, Berlin.

MILLER, G. L. 1984. Finding small simple cycle separators for 2-connected planar graphs. In *Proceedings of the 16th Symposium on Theory of Computing*. ACM, New York, 376–382.

NÄHER, S. AND MEHLHORN, K. 1999. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press. `http://www.algorithmic-solutions.com`.

OZKAN, S. B., WU, G. A., CHODERA, J. D., AND DILL, K. A. 2007. Protein folding by zipping and assembly. In *Proceedings of the National Academy of Sciences*. National Academy of Sciences, Washington, DC.

SPIELMAN, D. A. AND TENG, S.-H. 1996. Disk packings and planar separators. In *Proceedings of the 12th Annual Symposium on Computational Geometry*. ACM, New York, 349–358.

Received September 2008; revised May 2009; accepted June 2009