Engineering Planar Separator Algorithms *

Martin Holzer† Grigorios Prasinos‡ Frank Schulz† Dorothea Wagner†
Christos Zaroliagis‡

July 12, 2005

Abstract

We consider classical linear-time planar separator algorithms, determining for a given planar graph a small subset of the nodes whose removal separates the graph into two components of similar size. These algorithms are based upon *Planar Separator Theorem*s, which guarantee separators of size $O(\sqrt{n})$ and remaining components of size less than $2n/3$. In this work, we present a comprehensive experimental study of the algorithms applied to a large variety of graphs, where the main goal is to find separators that do not only satisfy upper bounds but also possess other desirable qualities with respect to separator size and component balance. We propose the usage of *fundamental cycles*, whose size is at most twice the diameter of the graph, as planar separators: 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.

1 Introduction

The *Planar Separator Theorem* was introduced by Lipton and Tarjan in [13], where they give a linear-time algorithm for determining a set of nodes (separator) of size smaller than $2\sqrt{2n} \approx 2.83\sqrt{n}$ that separates a given planar graph with n nodes into two components of size smaller than $2n/3$. Djidjev [7] improved the bound on the separator size to $\sqrt{6n} \approx 2.45\sqrt{n}$, and also proved a lower bound of $1.55\sqrt{n}$, which is still the best known. The algorithms behind these two classical results share a common core algorithm, 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, and the upper bound on separator size has been improved by Alon, Seymour and Thomas [2] to $2.13\sqrt{n}$ and by Djidjev and Venkatesan [8] to the currently best known bound of $1.97\sqrt{n}$ (where the aforementioned core algorithm is used as a subroutine, too). A recent work by Alexandrov et al. [1] considered a generalization of planar separator algorithms that computes t-separators: nodes have associated costs and weights, which are used in calculating the cost of the separator and the weights of the components resp.; the weight of each remaining component is required to be less than or equal to $t \cdot w(G)$, where t is an arbitrary constant in $(0, 1)$ and $w(G)$ the total weight of the graph. This typically requires the graph to be separated into more than two components, and with $t = 2/3$ and unit weight and cost includes the basic variant of the problem, as introduced above. The paper includes experimental study on a few synthetic and real-world families of graphs. For comparison, we consider some of the families used in [1] in our experiments.

We are not aware of a systematic and detailed experimental study regarding the classical algorithms by Lipton and Tarjan [13], and by Djidjev [7]. In this work, we do not only investigate finding separators that satisfy upper bounds, but we also consider several new algorithmic aspects regarding: (i) the optimization of separator size and balance; (ii) the consideration of fundamental-cycle separator algorithms in their own right; and (iii) the application of postprocessing techniques to improve the quality of the separators. The fundamental-cycle separator algorithms guarantee a bound on the separator size of $2d + 1$, where d

*This work was partially supported by the IST Programme of EC under contract no. IST-2002-001907 (DELIS).

†Department of Computer Science, University of Karlsruhe, P.O. Box 6980, 76128 Karlsruhe, Germany. Email: {mholzer,fschulz,dwagner}@ira.uka.de.

‡Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece, and Department of Computer Engineering and Informatics, University of Patras, 26500 Patras, Greece. Emails: {green,zaro}@ceid.upatras.gr.

denotes the diameter of a triangulation of the input graph. When the diameter is small, which is often the case for real-world graphs, this guarantees smaller separators than the $O(\sqrt{n})$ bounds of the classical algorithms.

Our main contribution in this work is the comprehensive experimental study of the above issues. It turned out that the behavior of the algorithms depends highly on the input graph: for example, on very regular graphs like grids a level of a breadth-first search tree (which is the first attempt of the classical algorithms) is already an almost optimal separator, whereas for more irregular and real-world graphs (e.g., road map graphs) the classical algorithms yield relatively bad solutions. Hence, for our experiments we used a large variety of planar graphs, both from real-world and synthetic inputs with different characteristics (e.g., size of diameter, size of minimum separator, etc). A surprising outcome of our experimental investigation is that fundamental-cycle separator algorithms always provide the best solutions. Another important issue of our experimental analysis concerns the arbitrary choices that have to be made during the course of an algorithm (e.g., the choice of a node as the root of a breadth-first search tree). It turns out that such choices influence the quality of the separators found significantly.

It should be noted that, although [1] deals with separators of planar graphs, the focus of our paper is different. We perform a more thorough experimental analysis regarding the classical algorithms, examining the interesting case of two balanced components and a small separator. We develop heuristics for improving the quality of the separation and the execution time. We use many kinds of graphs, both artificial and real-world (some of very large size), studying the effect of their characteristics on the execution time of the algorithm and on the heuristic techniques we employ.

We start (Section 2) with a brief review of the planar separator algorithms and their implementations, give our optimization criteria, and introduce the fundamental-cycle separator (FCS) algorithm. Moreover, we present the postprocessing techniques applied and address the issue of practically in the implementation of the FCS algorithm. Section 3 describes the graphs used for our experiments, while the results of our experimental study are reported in Section 4. We conclude in Section 5.

2 Separating Planar Graphs

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 $\beta\sqrt{n}$, for some constant β, on the separator size, while each of the remaining components contains less than two thirds of all nodes. The first theorem of this kind (for $\beta = 2\sqrt{2}$) and the Fundamental-Cycle Lemma, which constitute the foundation of our work, were introduced by Lipton and Tarjan [13]. For simplicity, we state the theorem and its related algorithms for the case of an unweighted planar graph. The algorithms and our implementations work for the weighted case, as it is introduced in [13], as well.

Theorem 1 (Planar Separator Theorem) *Given a planar graph G, the n nodes of G 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 no more than $\beta\sqrt{n}$ nodes, where β is a constant.*

An important concept used in the theorem are fundamental cycles: given a spanning tree of the input graph, a *fundamental cycle* consists of a non-tree edge e together with the path connecting the two end-nodes 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 consists of more than $2n/3$ nodes, and C is a fundamental cycle containing no more than $2h + 1$ nodes.*

2.1 Optimization Criteria

In practical applications of planar separator algorithms, requirements as to "good separations" may vary a lot. We therefore provide three optimization criteria: *separator size*, *balance*, and *separator ratio*. Let A be the smaller and B the larger of the two components, then *balance* is defined as A/B, and the *separator ratio* as S/A (cf. [12]). What is desirable are small separator size and high balance at the

same time; the separator ratio, which is to be minimized, represents a trade-off between the two targets. Note that if one of the simple criteria, separator size or balance, is to be optimized and there are several optimal solutions, then the separator ratio criterion becomes relevant.

2.2 The Algorithms

We investigate two classical algorithms, by Lipton and Tarjan (LT) [13] and by Djidjev (Dj) [7]. Both work in three phases. First, a breadth-first search (BFS) tree is computed, partitioning the nodes into levels. If one of the BFS levels constitutes a separator fulfilling the size and balance requirements, then the algorithm returns that level. In case there are several feasible levels and an optimization criterion is applied, the respective algorithm selects a level that is optimal with respect to that criterion. In the second phase, separators consisting of two levels of the BFS tree are considered, yielding a separation of the tree into a lower, middle, and upper part of the graph. If there is a separator such that the biggest of these parts and the remaining two put together each meet the bound, it is returned. If not so, the third phase applies Lemma 1 to one part of a previous two-level separation. In this case, the separator consists of those two levels and the fundamental cycle found through the lemma. The algorithms differ in the selection of the levels, as described in more detail below. We also consider fundamental-cycle separations computed by applying (the algorithmic version of) Lemma 1 directly to the graph.

Note that there are several parts in all the above algorithms, where certain arbitrary (in a sense "random") decisions have to be made: (i) the choice of the BFS root and the search itself; (ii) the triangulation of the graph, which is needed in phase 3 of the algorithms; (iii) the choice of the fundamental cycle from among several feasible ones—the so-called choice of the *non-tree edge*, as explained in Section 2.4. We will thoroughly discuss the influence of the choices of the BFS root and the non-tree edge in Section 4.

Lipton and Tarjan (LT). First, the middle level in the BFS tree is considered, i.e., the first level, starting from the root, that covers together with the lower levels more than half of the nodes. If this level is too large, the levels above and below are scanned until in each direction a level of size less than $2(\sqrt{n} - D)$ is found, where D is the distance to the middle level. If the part between these two levels is too large then Lemma 1 is used to separate it and in this case the separator consists of the two levels plus a fundamental cycle. We consider a textbook version [14, 11] of the algorithm guaranteeing a separator of size less than $4\sqrt{n}$, i.e., $\beta = 4$.

Djidjev (Dj). Already in [13], Lipton and Tarjan give an even better bound, and in [7] Djidjev further improves the selection of levels to $\beta = \sqrt{6} \approx 2.45$. In a similar but more sophisticated way than that in LT, the algorithm tries to find a separator consisting of one or two levels of the BFS tree (which have to be smaller than in LT), and as final option also determines a fundamental cycle.

Fundamental-Cycle Separation (FCS). During the experimental phase of this study, we observed that it is very effective to omit the selection of levels and directly consider fundamental cycles as separators: We compute a simple-cycle separator by applying Lemma 1 directly to the input graph. The height of any spanning BFS tree is smaller than the *diameter* d of the graph, and thus, for BFS trees, the fundamental cycle C computed by Lemma 1 is a separator with no more than $2d + 1$ nodes. The minimum height of a spanning tree equals the *radius* r of the graph, and in this case the fundamental cycle can be guaranteed to contain no more than $2r + 1$ nodes. A spanning tree of height r can be computed in time $O(n^2)$ simply by computing the breadth first search trees originating from every node in the graph. Hence, FCS computes, in linear time, a separator of size no more than $2d + 1$, and, by investing quadratic time, even a separator of size $2r + 1$ can be guaranteed.

Simple-cycle separators are also promising from a theoretical point of view, since an upper bound on the separator size of $1.97\sqrt{n}$, which is (to our knowledge) the best bound in n that is currently known [8], is achieved by a simple cycle.[1] For graphs of small diameter and radius, the FCS approach guarantees

[1] The algorithm used in the proof of the $1.97\sqrt{n}$ bound is rather sophisticated, and since the simple-cycle separators obtained by FCS are (often by far) smaller than this bound for all graphs we are considering, we restrict our experiments to FCS concerning simple-cycle separators.

a better bound than the $\beta\sqrt{n}$ bounds, whereas in general, of course, the $\beta\sqrt{n}$ bounds are stronger since the maximum diameter and radius of planar graphs are linear in the number of nodes.

Optimized versions. We have also implemented *optimized versions* of LT, Dj, and FCS that select an optimal separator according to a specific optimization criterion (cf. Section 2.1).

2.3 Postprocessing

To the above algorithms we provide two optional postprocessing steps, which may help to improve the separation found by the specific algorithm in terms of separator size and/or balance. The first one, called *node expulsion*, consists of moving separator nodes that are not connected to both components A and B (and hence do not actually separate two nodes from different components) to one of the components, thus decreasing the size of the separator. If a node can be moved to either component, then it is assigned to the smaller one, so that imbalance does not deteriorate. The idea behind the other postprocessing step, called the *Dulmage-Mendelsohn optimization* [3], is to 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 decreases the size of the separator and may also reduce imbalance. More precisely, S and the set of nodes in B adjacent to S, $Adj(S) \cap B$, are decomposed into S_I, S_X, and S_R and B_I, B_X, and B_R, respectively, forming the Dulmage-Mendelsohn decomposition. Setting $S' = S_I$ and $C = B_X = Adj(S_I) \cap B$ reduces the separator size if and only if $|S_I| - |B_X| > 0$, and this S' is one subset of S yielding maximum reduction. A similar fact holds for $S' = S_I \cup S_R$ and $C = B_X \cup B_R$, so that we pick from among these choices of S' that one yielding better balance. For details on the construction of S' we refer to [3].

2.4 Implementing the Fundamental-Cycle Lemma

The proof of Lemma 1 given by Lipton and Tarjan in [13] is a constructive proof in the sense that it delivers an algorithm for computing the desired fundamental cycle. This algorithm provides the core to all planar separator algorithms under investigation. Briefly, the algorithm, which will be referred as the *Fundamental-Cycle Separator (FCS) algorithm*, is as follows. The graph is triangulated and a spanning tree of the appropriate height is constructed. Every non-tree edge forms together with some tree edges a cycle. Each of these cycles is a candidate fundamental cycle separating the graph into two parts, so the goal is to find a cycle such that these two parts have the appropriate sizes. The optimized version of the FCS algorithm selects a cycle that induces an optimal separator according to a specific optimization criterion.

The FCS algorithm proceeds by examining each non-tree edge and the corresponding cycle, keeping track of the nodes on the cycle as well as of those inside it and their weight. Depending on whether the two edges, which form together with the current non-tree edge the triangular face lying *inside* the cycle, belong to the tree, the algorithm combines information computed for previous cycles until the desired cycle is found (e.g., in the case that one of the other two edges is non-tree and its corresponding cycle lies inside the current cycle). It is relatively straightforward to show that this strategy will compute a fundamental cycle.

Although the description of the algorithm is clear, there is a subtle problem of how to deal with the notions *inside* and *outside*. To compute correctly the information about cycles, there must be a sense of direction so that the non-tree edges are examined in a meaningful order. Our approach provides this sense of direction. We make use of the dual of the planar graph. The following well-known lemma provides an interesting property linking the spanning trees of a planar graph and its dual.

Lemma 2 *Let $G = (V, E)$ be a connected planar graph with dual $G^* = (V^*, E)$, and let $E' \subseteq E$. Then, $T = (V, E')$ is a spanning tree of G iff $T^* = (V^*, E - E')$ is a spanning tree of G^*.*

The given planar graph G is triangulated, a spanning tree $T = (V, E_T)$ of appropriate height is found using a simple breadth-first search, and the dual G^* is constructed. By Lemma 2, the edges $E - E_T$ form a spanning tree $T^* = (V^*, E - E_T)$ of G^*. A node of T^* is chosen (arbitrarily) to be the root of T^*

graph	nodes	edges	diameter		radius		LT		Dj		FCS	
			orig	triang	orig	triang	min	mean	min	mean	min	mean
`grid`	10000	19800	198	67	100	50	**82**	106	**82**	106	89	*99*
`rectangular`	10000	19480	518	20	260	10	**20**	27	**20**	27	**20**	*20*
`sixgrid`	9994	14733	513	22	257	11	**21**	28	**21**	28	**21**	*21*
`triangular`	5050	14850	99	45	66	34	**58**	83	**58**	83	**58**	*68*
`globe`	10002	20100	101	101	76	67	**100**	119	**100**	119	**100**	*106*
`t-sphere`	10242	30720	96	96	80	80	**160**	169	**160**	169	**160**	*164*
`diameter`	10000	29994	3333	3333	1667	1667	**3**	4	**3**	4	**3**	*3.3*
`del`	10000	25000	56	45	46	36	206	300	82	113	**65**	*75*
`del-max`	10000	29971	52	48	43	39	204	314	86	117	**74**	*79*
`leda`	9989	25000	18	15	11	8	76	216	7	31	**5**	*8*
`leda-max`	10000	29975	15	14	9	8	56	205	7	26	**6**	*10*
`c-grid`	10087	19904	212	72	106	36	38	78	38	78	**5**	*6.4*
`c-globe`	10090	20325	144	142	73	71	**4**	96	**4**	96	**4**	*12*
`c-del-max`	10005	29972	65	58	34	29	74	318	19	65	**5**	*8.3*
`c-leda-max`	10005	29984	20	16	11	8	78	209	7	32	**4**	*4.5*
`airfoil1`	4253	12289	65	31	36	21	50	89	**26**	85	**26**	*35*
`city2`	2948	3564	131	14	66	9	15	39	15	39	**4**	*9.5*
`city3`	15868	16690	658	13	329	9	28	53	28	53	**4**	*6.8*

Table 1: Large graphs: parameters and results. The table depicts the number of nodes and edges, the diameter and the radius for both the graph itself (orig) and a triangulation of it (triang) as well as for each of the algorithms LT, Dj, and FCS, all of them optimized on separator size *with* postprocessing applied, the minimum and mean separator sizes over all BFS root nodes; bold-face and italic figures indicate the best result(s) for the respective graph.

and all edges of T^* are directed away from this root. It can be easily proven that all these constructions require linear time.

Since there is a one-to-one correspondence between the non-tree edges in the original planar graph and the tree edges of the dual, there is a correspondence between the cycles in the original and the tree edges of the dual graph. By first examining the cycles corresponding to the edges that lead to the leaf nodes of T^* and continuing towards the root, the properties of all the cycles can be computed inductively. The direction of the edges in T^* ensures that when examining a cycle, all information needed (from cycles that lie inside it) will have been already computed.

Evidently, our strategy provides the necessary order of cycle examination. It should be noted that in the actual implementation of the algorithm the construction of the dual graph and its spanning tree can be avoided. Instead, the construction of the spanning tree is simulated by performing a "breadth-first" traversal on the *faces* of the original graph using the non-tree 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 Data Sets

In the following, we give a brief description of the graph classes we used in our experiments and are illustrated in Figure 1. Some parameters of the graphs that we used in the experiments (see Section 4) are reported in Table 1. The first five categories consist of synthetically generated graphs, whilst the data sets in the last stem from real world.

Grid-like graphs. This category encompasses three classes of regular-structured graphs, namely `grid`, `rect(angular)`, `sixgrid`, and `triang(ular)`. The `grid` and `rect` graphs can be regarded as an $x \times x$ or $x \times y$ raster of nodes, respectively, where adjacent nodes of the same row or column are connected by an edge. A `sixgrid` graph is composed of $x \times y$ hexagons in a honeycomb-like fashion, and a `triang` graph is generated iteratively as follows: starting with one triangle, in each iteration in the middle of every edge a new node is placed, and for each inner triangle three new edges are added that connect the

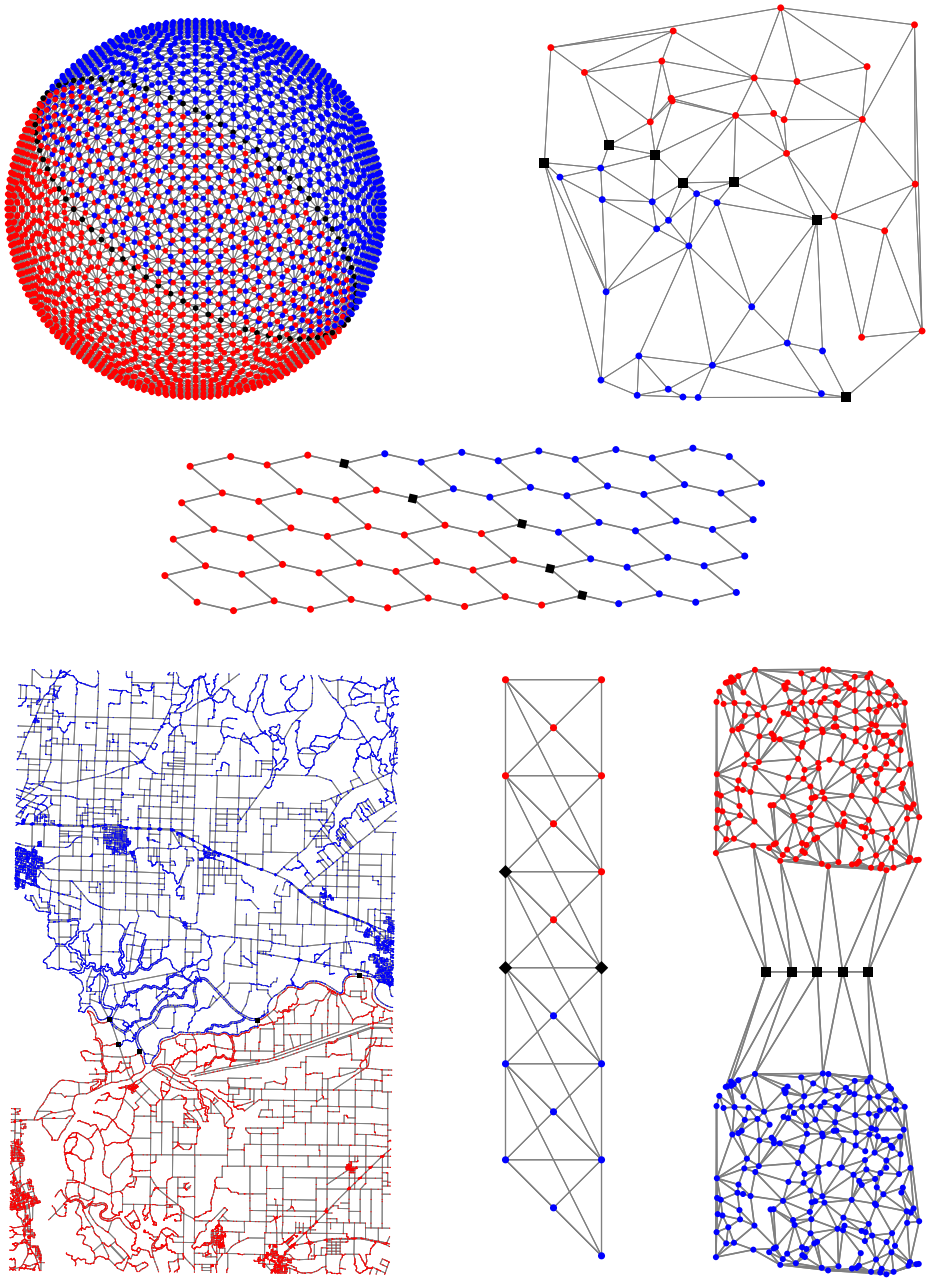

Figure 1: Sample graphs. From top to bottom and left to right: `t-sphere`, `del`, `sixgrid`, `city5`, `diameter`, and `c-del-max`. Nodes belonging to a component are drawn as red (grey) and blue (black) circles, separator nodes as black squares.

three new nodes on the edges of the respective triangle. In a `grid` graph with n nodes a separator with minimal size consists of approximately $\sqrt{2n/3} \approx 0.82\sqrt{n}$ nodes. If $x \ll y$, then the smallest separator of a `rectangular` graph has x nodes, and a `sixgrid` graph has an optimal separator with $x + 1$ nodes.

Sphere approximation. In [7] the currently best lower bound of $1.55\sqrt{n}$ on the separator size is proven by graphs that approximate the sphere. We consider two simple constructions of graphs that approximate the sphere, as *worst-case* examples concerning separator size. A `globe` graph is—simply speaking—the graph induced by (a specified number of) meridians and circles of latitude of a terrestrial globe. A `t-sphere` graph approximates the sphere by almost similar triangles, see e.g., [5]. The iterative generation process starts with an icosahedron (consisting of 20 equilateral triangles with all nodes on the sphere). During an iteration each triangle is split into four smaller ones.

Graph with big diameter. Given a diameter d, we construct a maximal planar graph that consists of $3d + 1$ nodes and has diameter d. We refer to this class as `diameter`. By construction, such a graph has always a separator of size 3.

Random planar graphs. Random maximum planar graphs, denoted by `del-max` and `leda-max`, are generated such that the specified number of nodes are randomly placed in the plane and the convex hull of them is triangulated, the triangulation being a Delaunay triangulation (`del-max`) or a standard LEDA-triangulation [15] (`leda-max`), respectively. In addition, we have the `del` and `leda` graphs, which are obtained from `del-max` and `leda-max`, respectively, by deleting at random a specified number of edges. We will occasionally refer to `del` and `del-max` (`leda` and `leda-max`, resp.) as the Delaunay (LEDA, resp.) graphs.

Graphs with small separators. Given a planar graph, two copies of this graph are connected via a given small number of additional nodes, which constitute a perfectly balanced separator of a so constructed graph. The challenge of the algorithms is to re-determine these small separators. We consider four of the previous graph types and get the following new kind of generated graphs: `c-grid`, `c-globe`, `c-del-max`, and `c-leda-max` graphs.

Real-world graphs. Regarding real-world data, we consider a graph representing a finite-element mesh [6] (`airfoil1`), and seven graphs representing the road networks of some U.S. cities and their surrounding areas (referred to as `city`), taken from the San Francisco Bay Area Regional Database (BARD) [4] and the Environmental Systems Research Institute (ESRI) info-page [9].

4 Experiments

Our experimental study is subdivided into three parts encompassing graphs of increasing size. The three algorithms LT, Dj, and FCS have been implemented in C++ using the LEDA library [15] (version 4.5). The code is compiled with GCC (version 3.3.3) and the experiments were performed on a 2.8 GHz Intel Xeon machine running a Linux kernel (version 2.6.5).

4.1 Small Graphs

For each of the generated graph types `grid`, `rect`, `sixgrid`, `globe`, `del`, `leda`, `del-max`, and `leda-max`, we considered series of 20 graphs containing between 50 and 1000 nodes. We take into account all algorithms, LT, Dj, and FCS, optimized on separator size, and each node was once chosen as BFS root. As already described above, if more than one smallest separators have been found, the one with best balance is selected.

Separator size and balance. Concerning the grid-like and `globe` graphs, the differences between the three algorithms are quite small. Due to the regular construction of these graphs, LT and Dj always succeed right after the first phase, and the smallest BFS level is almost optimum. The mean size of a fundamental-cycle separator is always slightly smaller and yields better balance. For the randomly

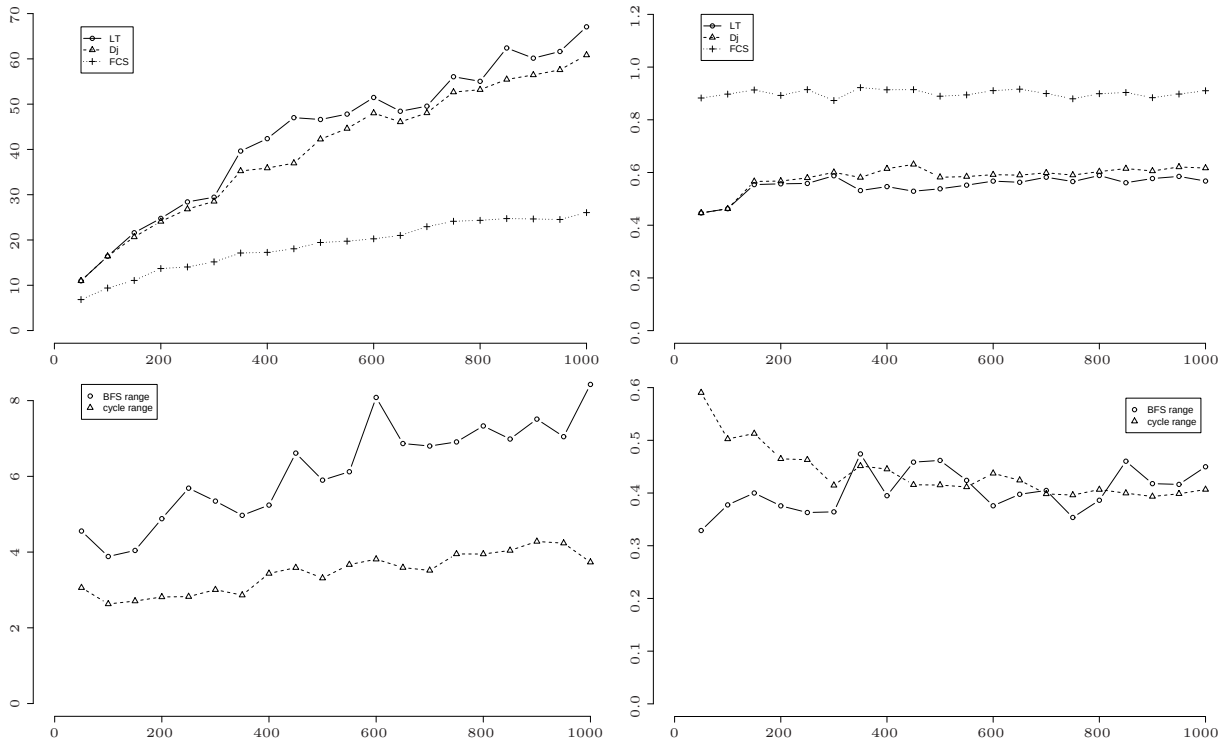

Figure 2: Experiments with a series of Delaunay graphs of sizes ranging from 50 to 1000 nodes: The upper diagrams show the mean separator size (top left) and mean balance (top right) with LT, Dj, and FCS. The lower diagrams show for FCS the ranges of the mean separator size (bottom left) and mean balance (bottom right), comparing BFS root and non-tree edge selection.

generated graphs, the results are different. Although for the Delaunay graphs, LT always terminates after the first phase with a smallest valid BFS level, Dj applies for around 15 per cent of the BFS roots the last phase of the algorithm.

In Figure 2, the diagrams in the upper row show the mean values of separator size and balance for the case of Delaunay graphs. It can be clearly seen that FCS computes on average the best separators, while Dj is slightly better than LT. Considering the LEDA random graphs, the results are again different: With those, both LT and Dj always have to pass the third phase. The mean separator size of FCS is only slightly better than that of Dj, while LT is by far worse. The mean balance with LEDA graphs is similar for all three algorithms, in the range between 0.8 and 0.9.

BFS root and non-tree edge selection. The lower diagrams in Figure 2 show the influence of BFS root selection and non-tree edge selection on separator size and balance. For FCS applied to the Delaunay graphs, the range of the mean of both the separator size and balance values are depicted, either over all possible BFS root nodes or over all possible non-tree edges. For example, the *range of the mean separator size over all possible BFS root nodes* is defined as follows: For every BFS root, determine the mean separator size over all possible non-tree edges. Then, the wanted *range of the mean* is the difference between the maximum and the minimum of these separator sizes among all BFS root nodes. The other ranges are defined similarly. The diagrams show that selection of the BFS root node is more decisive for the separator size than non-tree edge selection. Concerning balance, both selections are of similar importance.

4.2 Large Graphs

The second series of graphs that we experimented with, the *large graphs*, consists of 16 graphs of the categories mentioned in Section 3 of size roughly 10000 (except for the three real-world graphs, which have between 3000 and 16000 nodes). The `rectangular` graph represents a 20×500 raster, the `sixgrid`

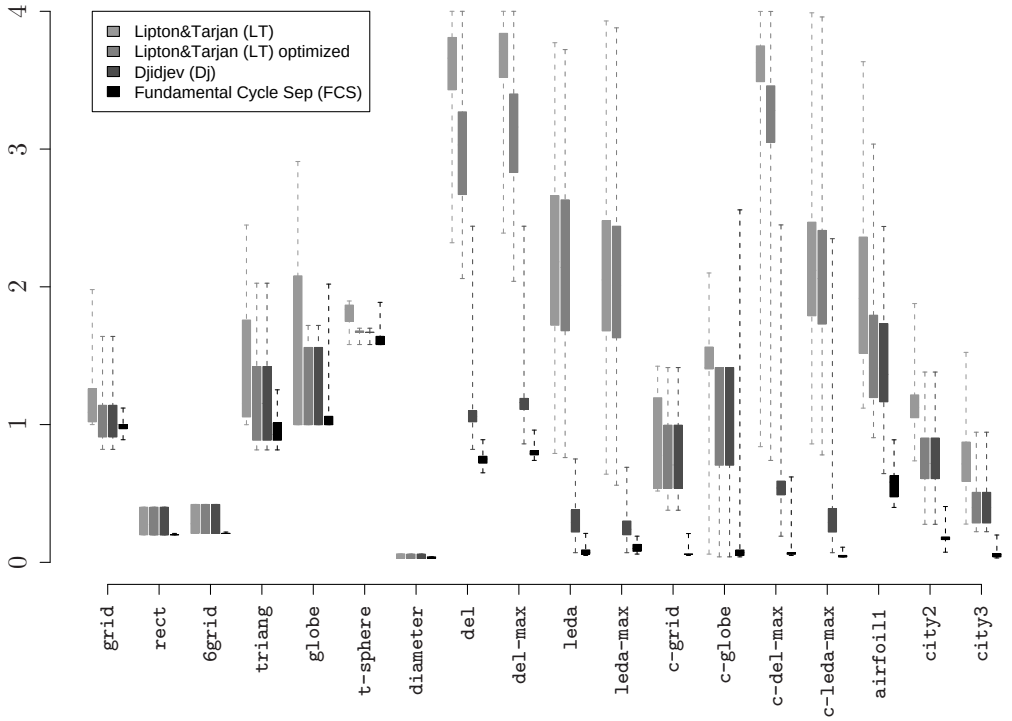

Figure 3: Box plots depicting the separator sizes relative to $\sqrt{n}$ obtained with unoptimized LT (light-gray) and LT (gray), Dj (dark-gray), and FCS (black), the latter three optimized on separator size. The dashed lines indicate the range of all separators *without* postprocessing applied.

graph consists of 20×237 hexagons, the `globe` has 100 meridians and circles of latitude, and the `t-sphere` is constructed by 5 iterations. For `c-grid`, `c-del-max` and `c-leda-max`, the two copies of the respective graph are connected by 5 nodes, while for `c-globe` only 4 nodes are used to connect the two `globe` graphs. A detailed synopsis of the graphs and some of their characteristics, such as the diameter and the radius, for both the original (orig) and the triangulated graph (triang), respectively, are reported in Table 1.

In the first experiments that we carried out for large graphs, we investigated the performance in terms of separator size of LT, both unoptimized and optimized on separator size, Dj, and FCS, the latter ones also optimized on separator size. We ran each of these algorithms for each graph while once making each node the root of the BFS tree.

The subsequent experiments deal with the effect of postprocessing on the various algorithms. In order to appropriately choose the postprocessing variant to be performed with each optimization criterion, we undertook a pre-study, performing for each graph and each combination of optimization and preprocessing steps one run of each of the above algorithms.

To get an idea of the quality of the separators found by the algorithms, we compare them against separators obtained with the help of *MeTiS* [10], a graph partitioning tool collection. Separators determined by MeTiS are a trade-off between separator size and balance, so for the sake of a meaningful comparison, we contrast the MeTiS results and our algorithms optimized on separator ratio.

Pre-study. Figure 4 depicts average values of the separator reduction relative to the former separator size and the absolute improvement in terms of balance and of separator ratio. These results suggest that optimization of separator size and separator ratio should be accompanied by a combination of Dulmage-Mendelsohn optimization as the first postprocessing step and node expulsion afterwards; the same holds when not optimizing at all. With balance as optimization criterion, no postprocessing step is on the average able to improve balance further.

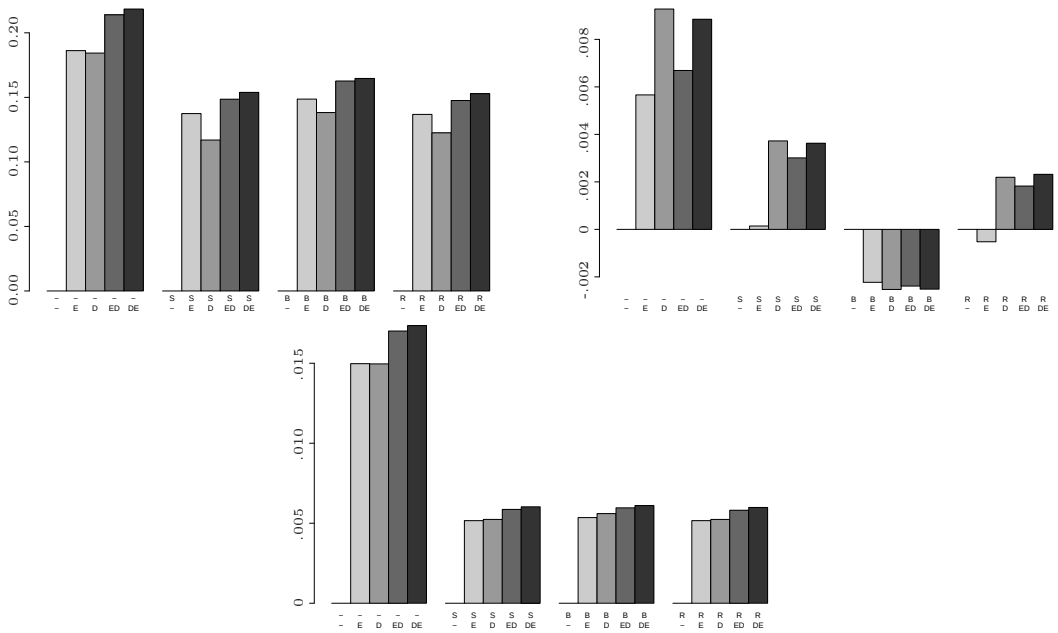

Figure 4: Pre-study: average separator reduction, balance improvement, and separator ratio improvement. The upper key in the x-axis labels denotes optimization (-: none, S: separator, B: balance, R: separator ratio); the lower one stands for postprocessing (-: none, E: node expulsion, D: Dulmage-Mendelsohn decomposition), where double letters reflect the application order of the postprocessing steps.

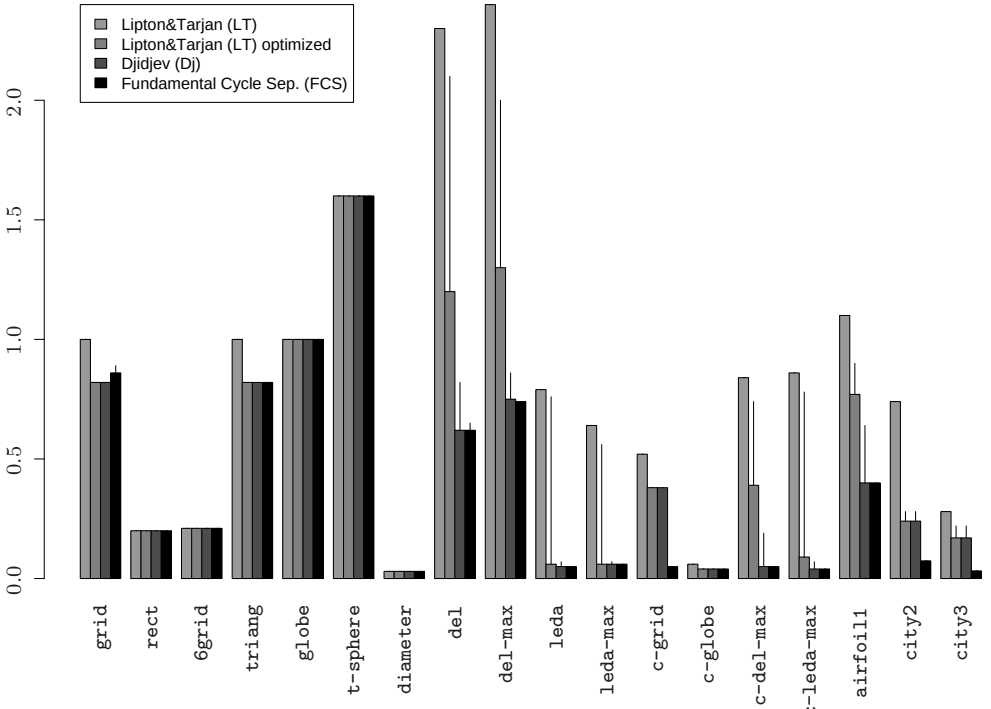

Figure 5: Postprocessing: average separator size for unoptimized LT (light-gray) and LT (gray), Dj (dark-gray), and FCS (black), the latter three optimized on separator size. The line on top of a bar shows the respective value before the postprocessing.

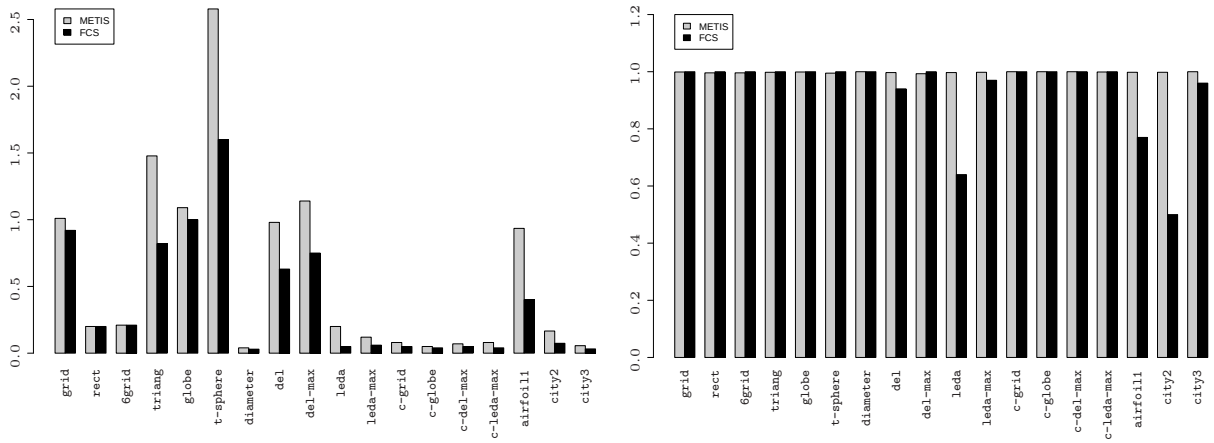

Figure 6: Minimum separator sizes relative to $\sqrt{n}$ (left) and balance (right) computed with MeTiS (light-gray) and with FCS (black) optimized on separator ratio.

Main results. The results of the experiments regarding the separator sizes achieved by the various algorithms are listed in Table 1, and illustrated in Figure 3 by means of box plots that represent the middle fifty per cent of the data series (note that the whiskers here span the whole range of outcomes). The data shows that—except for the `grid` graphs—the smallest minimum separator is found by FCS, and concerning the mean separator size FCS achieves the best result for all graphs under consideration. This, together with the fact that the boxes are clearly slender, and—except for `c-globe`—the ranges are minimal for FCS, suggests that FCS significantly outperforms the other algorithms in terms of separator size. In particular, this behavior is surprising for graphs with rather big diameter d and radius r (e.g., `c-globe`, `globe`, and `diameter`), since the guaranteed bound on the separator size is $2d + 1$ ($2r + 1$, respectively; cf. the description of FCS on page 3) for FCS.

For regular-structured graphs (i.e., grid-like, sphere approximation, and the `diameter` graphs) the separator sizes are similar and quite high for the three algorithms. For irregular graphs (i.e., `leda`, the graphs with small separator, and the real-world graphs), the picture looks different: The Dj algorithm always yields better results than the LT algorithm, and FCS is clearly superior to both Dj and LT. Furthermore, the minimum and mean separator sizes computed by FCS are by far below the guaranteed upper bounds.

The running time considering one BFS root node is linear for all of our algorithms. However, for the algorithms LT and Dj, the constant crucially depends on the phase in which the algorithms terminate (cf., Section 2.2). The first two phases consist basically of a breadth-first search, while the computation of the fundamental cycle requires expensive operations like embedding, triangulation, and copying. FCS, of course, computes a fundamental cycle and always needs the expensive operations. LT and Dj terminate after phase 1 with all grid-like graphs, sphere-approximating graphs, and with the `diameter`, `c-grid`, `c-globe`, and `city` graphs. In contrast, the LEDA, `c-del-max`, and `c-leda-max` graphs in the majority of cases require phase 3. For the Delaunay graphs, LT mostly terminates after phase 1, but Dj needs phase 3. The mean running time for LT (applying only phase 1) in the `city3` graph, for example, is 0.04 seconds, while FCS involving a fundamental-cycle computation needs 0.71 seconds.

Postprocessing. Figure 5 depicts the average separator size achieved with the diverse algorithms before and after the application of a postprocessing step. On the one hand, for the grid-like and sphere-approximating graphs as well as the `diameter` graph, the separators found by the algorithms without postprocessing cannot be much improved. (Often the separators are already close to optimal solutions for these graphs.) On the other hand, for the remaining graphs, the separators computed by the algorithms Dj and LT are very large compared to an optimal solution, and in these cases the postprocessing greatly improves the separators. The separators computed by FCS can generally be improved only a little.

Benchmark. MeTiS provides—amongst others—implementations of two algorithms, *kmetis* and *pmetis*, for computing small-cardinality k-way edge partitionings with balancing constraints. The application of

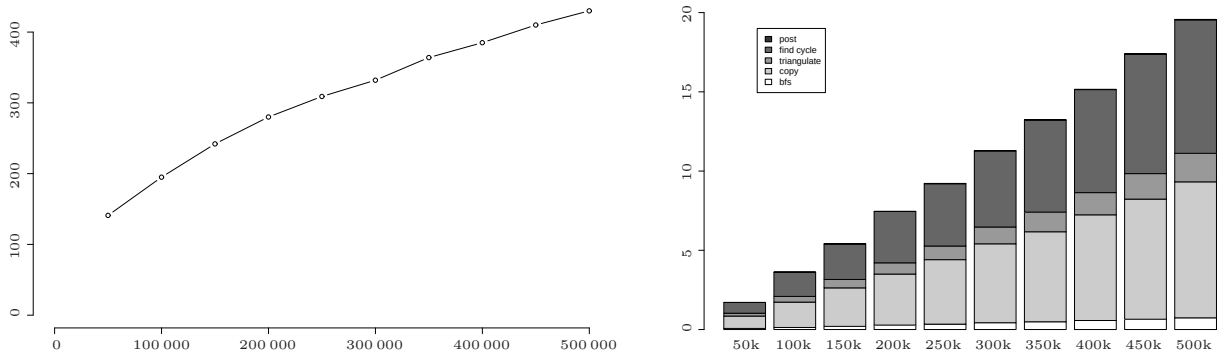

Figure 7: Delaunay graph series: Separator size (left) and computation time in seconds (right). The separators were computed with FCS for ten randomly selected BFS roots optimized on separator ratio. Shares in computation time for one BFS root (from bottom to top): computing the BFS tree, copying the graph, triangulating the graph, determining fundamental cycles, postprocessing.

both *kmetis* and *pmetis* on all of our instances yield 2-partitionings of very high balances (meeting the requirement that each part encompass at least one third of the graph's nodes) with quite few cut edges.

From an edge partitioning thus obtained we can get a node separator by choosing an appropriate subset of the end-nodes of the edges forming the cut. To achieve this, our implementation proceeds as follows: at each time, until all cut edges are covered, i.e., have one of their end-nodes included in the separator, we pick one end-node as a separator node, trying to cover with it as many cut edges not covered yet as possible.

Since among LT, Dj, and FCS optimized on separator ratio, the solutions computed by FCS were the best with respect to both separator size and balance, we compare MeTiS with FCS only. Figure 6 shows the best separator size and balance values. One may state that with FCS, the separator size is always at least as good as with MeTiS and balance is almost always comparable. Those graphs whose balance is considerably worse with FCS than with MeTiS (`leda` and `city2`) exhibit by far smaller separators with FCS, which suggests that the weighting between the two criteria, separator size and balance, seems to be more in favor of separator size with FCS, while MeTiS tends to prefer balance. Indeed, almost perfect balance can always be achieved with FCS optimized only on balance.

4.3 Very Large Graphs

Under the name of very large graphs, we consider two series of graphs increasing in size: a series of city graphs with numbers of nodes up to about 45,000 and a series of ten random `del` graphs of sizes between 50,000 and 500,000. For these graphs we computed separations by the following linear-time procedure: run FCS on ten BFS trees of a given graph, determined by a random node as root, and from among these separations take the one with best separator ratio.

The results of the experiments with the city graph series are depicted in the table aside. Obviously, all city graphs have extremely small separators, which are also found by our algorithm. The separators for the `city2` and `city3` graphs, which had already been included in the experiments of the previous section, are somewhat bigger than those of the preceding experiment (8 and 7 instead of 4, resp.; see Table 1). This is due to the fact that: (i) the separator ratio is now optimized instead of the separator size; and (ii) we do not longer take into account every node as a BFS root.

graph	nodes	edges	size	balance
`city1`	1429	3034	5	0.871
`city2`	2948	3564	8	0.996
`city3`	15868	16690	7	0.869
`city4`	20036	41476	10	0.789
`city5`	24106	53826	5	0.740
`city6`	38823	79988	8	0.704
`city7`	44878	90930	7	0.547

Figure 7 shows the separator sizes and the running times with the Delaunay graph series. The balance is very high for all graphs, namely greater than 0.94, and the separator size divided by $\sqrt{n}$ is always in the range between 0.6 and 0.63. This suggests that the separator ratio criterion indeed represents a good trade-off between separator size and balance. The running times vary between approximately 2 and 20 seconds. Also, the diagram pronouncedly reflects linearity of the algorithms' time complexity.

5 Conclusions and Outlook

Our experiments have shown that, especially for graphs with small separators, there is a high potential for optimizing the separators computed by the algorithms. Both the postprocessing and in particular the Fundamental-Cycle Separation yielded almost-optimal separators with respect to separator size and balance. Applied to graphs whose triangulations have small diameter (which is true for many graphs, especially from real world), FCS is empirically and theoretically superior to the classical algorithms guaranteeing separators of size $O(\sqrt{n})$. Selection of the non-tree edge in the fundamental-cycle computation has a considerable influence on both criteria, and we are able to select the best during the respective algorithm. The choice of the BFS root also exhibits a great impact on separator quality, mainly on its size. The experiments on city graphs and very large Delaunay graphs confirmed that FCS, applied to a small random sample of BFS root nodes and separator ratio as optimization criterion yields excellent separators in linear time.

An issue for further investigation would be to explore whether more sophisticated strategies for selecting an appropriate BFS root can be developed. In addition, we would like to investigate other parts of the algorithms that are also subject to arbitrary choices, namely triangulation and breadth-first search.

Acknowledgments

The authors would like to thank Imen Borgi and Jürgen Graf for their assistance with parts of the implementation work and the anonymous referees of ESA 2005 for their detailed comments and very helpful hints for further research.

References

[1] L. Aleksandrov, H. N. Djidjev, H. Guo, and A. Maheshwari. Partitioning planar graphs with costs and weights. In *ALENEX 2002*, volume 2409 of *LNCS*, pages 98–110. Springer, 2002.

[2] N. Alon, P. Seymour, and R. Thomas. Planar separators. *SIAM Journal on Discrete Mathematics*, 7(2):184–193, 1994.

[3] C. Ashcraft and J. W. H. Liu. Applications of the Dulmage-Mendelsohn decomposition and network flow to graph bisection improvement. Technical Report CS-96-05, Dept. of Computer Science, York University, North York, Ontario, Canada, 1996. `http://www.cs.yorku.ca/techreports/1996/CS-96-05.html`.

[4] BARD. Bay Area Regional Database. `http://bard.wr.usgs.gov`.

[5] P. Bourke. Sphere generation, 1992. `http://astronomy.swin.edu.au/~pbourke/modelling/sphere/`.

[6] R. Diekmann. Graph Partitioning Graph Collection. `http://wwwcs.upb.de/fachbereich/AG/monien/RESEARCH/PART/graphs.html`.

[7] H. N. Djidjev. On the problem of partitioning planar graphs. *SIAM Journal on Algebraic and Discrete Methods*, 3(2):229–240, 1982.

[8] H. N. Djidjev and S. M. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 34:231–243, 1997.

[9] ESRI. Environmental Systems Research Institute. `http://www.esri.com`.

[10] G. Karypis. MeTiS. `http://www-users.cs.umn.edu/~karypis/metis`.

[11] D. Kozen. *The Design and Analysis of Algorithms*. Springer, 1992.

[12] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

[13] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[14] K. Mehlhorn. *Data Structures and Algorithms 1, 2, and 3*. Springer, 1984.

[15] S. Näher and K. Mehlhorn. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999. `http://www.algorithmic-solutions.com`.