I/O-Optimal Algorithms for Planar Graphs Using Separators

Anil Maheshwari* Norbert Zeh*

Abstract

We present I/O-optimal algorithms for several fundamental problems on planar graphs. Our main contribution is an I/O-efficient algorithm for computing a small vertex separator of an unweighted planar graph. This algorithm is superior to all existing external memory algorithms for this problem, as it requires neither a breadth-first search tree nor an embedding of the graph as part of the input. In fact, we derive I/O-optimal algorithms for planar embedding, breadth-first search, depth-first search, single source shortest paths, and computing weighted separators of planar graphs from our unweighted separator algorithm.

1 Introduction

I/O-efficient graph algorithms have received considerable attention because massive graphs arise naturally in many applications such as web modeling, geographic information systems, or modeling of large communication networks. When working with massive data sets, the transfer of data between internal and external memory, and not the internal memory computation, is often the bottleneck. Thus, I/O-efficient algorithms can lead to considerable run-time improvements.

Planar graphs are a natural abstraction of many real world problems. They are among the fundamental combinatorial structures used in algorithmic graph theory; separators have played the key role in designing divide-and-conquer algorithms for planar graphs. The classical separator theorem for planar graphs of [15], coupled with linear-time embedding algorithms, led to phenomenal developments in algorithmic graph theory. There are numerous results on computing a variety of separators, as well as on applications of separators in various fields. These applications include lower bounds on the size of boolean circuits, approximation algorithms for NP-complete problems, nested dissection of sparse systems of linear equations, load balancing in parallel numerical simulations based on the finite elements method, partitioning triangular irregular networks in the field of GIS, and encoding graphs.

Breadth-first search (BFS) and depth-first search (DFS) are the two most fundamental graph searching strategies. They are extensively used in internal memory algorithms, as they are easy to perform in linear time; yet they provide valuable information about the structure of the given graph. However, it is not known how to perform BFS and DFS on arbitrary sparse graphs I/O-efficiently. These are among the major open problems [20]. This implies in particular that existing, elegant, algorithms for computing separators of planar graphs are not I/O-efficient, as they use BFS to partition the vertices into levels around some root vertex and then judiciously use these levels to compute the separator.

1.1 Model of Computation

The algorithms in this paper are designed and analyzed in the Parallel Disk Model (PDM) [21]. In this model, D identical disks of unlimited size are attached to a machine with an internal memory capable of holding M data items. These disks constitute the external memory of the machine. Each disk is partitioned into blocks of B data items each. An I/O-operation is the transfer of up to D blocks, at most one per disk, to or from internal memory from or to external memory. The complexity of an algorithm is the number of I/O-operations it performs. Sorting, permuting, and scanning a sequence of N consecutive data items take $\mathrm{sort}(N) = \Theta((N/(DB)) \log_{M/B}(N/B))$, $\mathrm{perm}(N) = \Theta(\min(N, \mathrm{sort}(N)))$, and $\mathrm{scan}(N) = \Theta(N/(DB))$ I/Os, respectively [21].

1.2 Previous Results

A number of I/O-efficient graph algorithms [1, 2, 5, 6, 8, 12, 14, 16, 17, 18, 19] have been developed in recent years. We only discuss results on undirected BFS, DFS, single source shortest paths (SSSP), embedding, and graph separators here. The best known SSSP-algorithm for arbitrary undirected graphs takes $O(|V| + (|E|/B)\log_2|E|)$ I/Os [14]. The best known BFS-algorithm for arbitrary undirected graphs takes $O(|V| + \mathrm{sort}(|E|))$ I/Os [19]. Recently, a BFS-algorithm for graphs of bounded degree has been presented in [18]. If d is the maximum vertex degree

*School of Computer Science, Carleton University, Ottawa, K1S 5B6, Canada. Email: {maheshwa,nzeh}@scs.carleton.ca.
Research supported by NSERC and NCE GEOIDE.

in the graph, the algorithm takes $O(|V|/(\gamma \log_d B) + \text{sort}(B^\gamma |V|))$ I/Os using $O(|V|/B^{1-\gamma})$ blocks of external memory, for $0 < \gamma \leq \frac{1}{2}$. The algorithm can be combined with results from [5] to obtain an SSSP-algorithm for embedded planar graphs which takes $O(|V|/(\gamma \log_3 B) + \text{sort}(B^\gamma |V|))$ I/Os [18].

In [12], an $O(\text{sort}(N))$ I/O algorithm for computing a 2/3-separator of size $O(\sqrt{N})$ for an embedded planar graph with N vertices is given, provided that a BFS-tree is part of the input. In [5], this idea has been extended to obtain an $O(\text{sort}(N))$ I/O algorithm to compute a small ϵ-separator of an embedded planar graph, again provided that a BFS-tree of the graph is given. Using the computed separator, the SSSP problem can then be solved in $O(\text{sort}(N))$ I/Os [5]. In a recent paper [6], two DFS-algorithms for embedded planar graphs are presented. The first one takes $O(\text{sort}(N) \log_2 N)$ I/Os. The second one takes $O(\mathcal{I}(N))$ I/Os, where $\mathcal{I}(N)$ is the number of I/Os required to compute a BFS-tree of an embedded planar graph.

In [16, 17], $O(\text{sort}(N))$ I/O algorithms for BFS, DFS, and SSSP on outerplanar graphs and graphs of bounded treewidth have been presented. It is shown in [16] that outerplanar embedding requires $\Omega(\text{perm}(N))$ I/Os. The same lower bound for BFS, DFS, and SSSP follows from the $\Omega(\text{perm}(N))$ I/O lower bound for list-ranking [8].

In internal memory, the problems of computing embeddings and separators for planar graphs are well-studied. We mention the most relevant results here. In [15], it is shown that a $\frac{2}{3}$-separator of size $O(\sqrt{N})$ for a given embedded planar graph G of size N can be computed in linear time. In [9], the algorithm of [15] is applied recursively to compute in $O(N \log N)$ time a set S of $O(N/\sqrt{h})$ vertices whose removal partitions G into $O(N/h)$ (possibly disconnected) subgraphs of size $O(h)$ each of which is adjacent to at most $O(\sqrt{h})$ vertices in S. In [3], it is shown how to compute a set S of $O(\sqrt{(g+1/\epsilon)N})$ vertices whose removal partitions an embedded graph G of genus g into subgraphs of size at most ϵN. Other results include results on edge separators, separators for graphs with multiple vertex weights, and separators of low cost if a cost function of the vertices of G is given. All these separator algorithms use BFS to investigate the structure of the graph. An embedding of a planar graph can be computed in linear time, using for instance one of the algorithms of [11, 7]. In [13], it is shown that the SSSP problem can be solved in linear time on planar graphs with non-negative edge weights.

1.3 New Results We show how to find a separator S of size $O(N/\sqrt{h})$ whose removal partitions a given planar graph G into $O(N/h)$ subgraphs, each of size at most h, so that each subgraph is adjacent to at most $\sqrt{h}$ separator vertices. Our algorithm takes $O(\text{sort}(N))$ I/Os, provided that $M \geq 56h \log^2(DB)$. In contrast to previous results, our algorithm requires neither an embedding nor a BFS-tree of G to be given as part of the input. In fact, we use our new separator algorithm to develop an $O(\text{sort}(N))$ I/O algorithm for planarity testing and planar embedding. Given a separator S and a planar embedding of the graph, results of [5, 6] can be applied to obtain $O(\text{sort}(N))$ I/O algorithms for BFS, DFS, and shortest paths on planar graphs. Due to the lack of space we sketch the main ideas here and refer the reader to [23, 24] for details.

Our separator algorithm can be applied to any class of sparse graphs (i.e., graphs of bounded arboricity) having small separators, provided that an efficient internal memory algorithm for computing separators is known for that class of graphs. The proof of [16] can easily be adapted to show that planar embedding requires $\Omega(\text{perm}(N))$ I/Os (see [23]). Our algorithms for embedding, BFS, DFS, and SSSP can be made to run in $O(\text{perm}(N))$ I/Os, thereby matching the lower bounds, by running our algorithms and the linear-time internal memory algorithms simultaneously, and stopping as soon as one the of two algorithms finishes.

In the next subsection, we present some results that are used in our separator and embedding algorithms. Section 2 presents our separator algorithm. Section 3 presents an outline of our algorithm for planarity testing and planar embedding. In Section 4, we briefly discuss how these two results can be applied to obtain I/O-optimal algorithms for BFS, DFS, single source shortest paths, and weighted ϵ-separators on planar graphs.

1.4 Preliminaries Given a set $S \subseteq V$ of vertices of a graph $G = (V, E)$ and a subgraph $H \subseteq G - S$, the *boundary* of H, denoted by ∂H, is the set of vertices in S adjacent to H. The following results will be applied in our separator algorithm.

THEOREM 1.1. [3] *Given a planar graph $G = (V, E)$ and an integer $h > 0$, it takes $O(N)$ time to compute a set $S \subseteq V$ of $O(N/\sqrt{h})$ vertices so that no connected component of $G - S$ has size exceeding h.*

THEOREM 1.2. [9] *Given a planar graph $G = (V, E)$ and a set $S \subseteq V$ of vertices whose removal partitions G into $O(N/h)$ subgraphs $H_1, \ldots, H_q$ such that $|H_i| \leq h$ and $\sum_{i=1}^{q} |\partial H_i| = O(N/\sqrt{h})$, it takes $O(N \log N)$ time to compute a set S' of $O(N/\sqrt{h})$ vertices, $S \subseteq S' \subseteq V$, whose removal partitions G into $O(N/h)$ subgraphs $H'_1, \ldots, H'_r$ such that $|H'_i| \leq h$ and $|\partial H'_i| \leq c\sqrt{h}$, for $1 \leq i \leq r$ and some constant $c \geq 0$.*

LEMMA 1.1. *Let $G = (V_1, V_2, E)$ be a simple connected bipartite planar graph such that the vertices in V_2 have degree at least three each. Then $|V_2| \leq 2|V_1|$.*

Proof. Consider an embedding $\hat{G}$ of G. As G is bipartite, every face of $\hat{G}$ has at least 4 edges on its boundary. Thus, $|F| \leq |E|/2$. By Euler's formula, $2 = |V| + |F| - |E| \leq |V| - |E|/2$, so that $|E| \leq 2|V|$. On the other hand, $|E| \geq 3|V_2|$, so that $3|V_2| \leq 2|V| = 2(|V_1| + |V_2|)$. This implies that $|V_2| \leq 2|V_1|$. □

COROLLARY 1.1. *Let $G = (V_1, V_2, E)$ be a simple connected bipartite planar graph. Let the vertices in V_2 be partitioned into equivalence classes $C_1, \ldots, C_q$, where two vertices v and w are equivalent if they have degree at most two and are adjacent to the same set of vertices in V_1. Then $q \leq 6|V_1|$.*

Given a graph $G = (V, E)$ and an edge $\{v, w\} \in E$, the *contraction* of edge $\{v, w\}$ is the operation of removing w from G and making all edges $\{u, w\} \in G$, $u \neq v$, incident to v. If the vertices of G have weights, then the weight of v is increased by the weight of w.

2 Separators for Planar Graphs

2.1 Outline The basic structure of our separator algorithm is not unlike that of the algorithm used in [13] to derive a hierarchical separator decomposition of the given graph, which recursively compresses the given graph and then refines the current partition as the compression steps are undone. However, the separators obtained by the algorithm of [13] are not optimal. We choose the parameters used in the compression steps and in computing the intermediate separators so that we obtain an optimal separator, and show how to perform each compression step I/O-efficiently.

The main idea of our I/O-efficient separator algorithm is to construct a hierarchy of $\log_2(DB)$ planar graphs $G_0, \ldots, G_r$ such that $G = G_0$, $|G_{i+1}| \leq \frac{1}{2}|G_i|$, and every vertex in G_{i+1} represents a small subgraph of G_i and thus a subgraph of G. Given this hierarchy, compute a small separator S_r partitioning G_r into relatively coarse subgraphs such that the number of vertices in G corresponding to the vertices in S_r is small. Then iteratively refine graph G_r to graphs $G_{r-1}, G_{r-2}, \ldots, G_0$ along with their partitions. The result is a separator of size $O(N/\sqrt{h})$ partitioning G into subgraphs of size at most $h\log^2(DB)$. Given such a partition, load each subgraph into internal memory and refine it so that no subgraph has size exceeding h or boundary size exceeding $\sqrt{h}$. This can be done using Theorems 1.1 and 1.2 and introduces at most $O(N/\sqrt{h})$ additional separator vertices. Thus, we obtain a separator of size $O(N/\sqrt{h})$ partitioning G into subgraphs of size at most h and with boundary size at most $\sqrt{h}$. If every recursive step can be realized in $O(\text{sort}(|G_i|))$ I/Os, our algorithm takes $O(\text{sort}(N))$ I/Os to compute the desired separator.

2.2 The Graph Hierarchy The first step is to compute the sequence of graphs $G_0, \ldots, G_r$. We first describe the construction of graph G_{i+1} from graph G_i and then prove a number of properties of graphs $G_0, \ldots, G_r$.

Given graph G_i, every vertex $v \in G_i$ has a *weight* $\omega(v)$, which is the number of vertices in G represented by v. That is, $\omega(v) = 1$, for every vertex $v \in G_0$, as $G_0 = G$. The *size* $\sigma(v)$ of a vertex $v \in G_i$ is the number of vertices in G_{i-1} represented by v. We define a series of weight thresholds $\rho_i = 2^{i+1}$, for $0 \leq i \leq r$. Given G_i, let $G'_{i+1} = G_i$ with $\sigma(v) = 1$, for all $v \in G'_{i+1}$. The weight of a vertex in G'_{i+1} is the same as its weight in G_i. As long as there is an edge $\{v, w\} \in G'_{i+1}$ such that $\omega(v) + \omega(w) \leq \rho_{i+1}$ and $\sigma(v) + \sigma(w) \leq 56$, contract edge $\{v, w\}$ and repeat. Let G''_{i+1} be the resulting graph.

A vertex v in G''_{i+1} is *heavy* if $\omega(v) \geq \rho_{i+1}/2$ or $\sigma(v) \geq 28$, and *light* otherwise. For every edge $\{v, w\} \in G''_{i+1}$, either $\omega(v) + \omega(w) > \rho_{i+1}$ or $\sigma(v) + \sigma(w) > 56$, so that at least one of v and w is heavy. Thus, no two light vertices are adjacent. Partition the light vertices of degree at most 2 in G''_{i+1} into classes $C_1, \ldots, C_q$ such that the vertices in each class have the same set of (heavy) neighbors. Further partition each class C_j into a minimal number of subclasses $C_{j,1}, \ldots, C_{j,k_j}$ so that the total weight of each subclass $C_{j,l}$, $1 \leq l \leq k_j$, is at most ρ_{i+1} and the total size of its vertices is at most 56.

Graph G_{i+1} is defined as follows: The vertex set of G_{i+1} consists of all heavy vertices of G''_{i+1}, all light vertices of degree at least 3, as well as one vertex $v_{j,l}$ for each class $C_{j,l}$. The vertices in G_{i+1} are assigned the following weights: The weight of a heavy vertex in G_{i+1} is the same as in G''_{i+1}. The same is true for all light vertices of degree at least 3. For every vertex $v_{j,l}$, let $\omega(v_{j,l}) = \sum_{v \in C_{j,l}} \omega(v)$. There is an edge between two vertices v and w in G_{i+1} if (i) they are adjacent in G''_{i+1} and heavy or of degree at least 3, or (ii) vertex v is heavy, $w = v_{j,l}$, and the vertices in $C_{j,l}$ are adjacent to v in G''_{i+1}.

LEMMA 2.1. *Let G be a planar graph, and let $G = G_0, \ldots, G_r$ be the graphs as defined above. Every graph G_i, $0 \leq i \leq r$, has the following properties:*

(i) Graph G_i is planar,

(ii) $\omega(v) \leq \rho_i$, for all vertices $v \in G_i$,

(iii) Every vertex $v \in G_i$ corresponds to at most 56 vertices in G_{i-1}, for $i > 0$, and

(iv) $|G_i| \leq 28N/\rho_i$.

Proof. Properties (i)–(iii) follow immediately from the construction of graphs $G_0, \ldots, G_r$. In order to show Property (iv), let h_i be the number of heavy vertices in G_i. It follows from the construction and Corollary 1.1 that $|G_i| \leq 7h_i$. Thus, Property (iv) follows if we can show that $h_i \leq 4N/\rho_i$.

We prove this claim by induction. We partition the heavy vertices into two categories. Heavy vertices of type I are heavy because their weight is at least $\rho_i/2$. Type-II vertices are heavy because their size is at least 28. In general, graph G_i contains at most $2N/\rho_i$ type-I vertices and at most $|G_{i-1}|/28$ type-II vertices. That is $h_i \leq \frac{2N}{\rho_i} + \frac{|G_{i-1}|}{28}$. For $i = 1$, we obtain $h_1 \leq \frac{2N}{\rho_1} + \frac{N}{28} < \frac{4N}{\rho_1}$ because $\rho_1 = 4$. For $i > 1$, we obtain $h_i \leq \frac{2N}{\rho_i} + \frac{|G_{i-1}|}{28} \leq \frac{2N}{\rho_i} + \frac{h_{i-1}}{4} \leq \frac{2N}{\rho_i} + \frac{N}{\rho_{i-1}} = \frac{4N}{\rho_i}$. □

2.3 The I/O-Complexity of Computing the Graph Hierarchy Next we show how to compute graphs $G = G_0, G_1, \ldots, G_r$ I/O-efficiently. Graph G_i is computed from G_{i-1} as follows: Initially, let $G'_i = G_{i-1}$. An edge $\{v, w\}$ of G'_i is called *contractible* if $\omega(v) + \omega(w) \leq \rho_i$ and $\sigma(v) + \sigma(w) \leq 56$. The contractible subgraph H_0 of G'_i is the graph induced by all contractible edges in G'_i. H_0 can easily be extracted from G'_i in $O(\mathrm{sort}(|G_{i-1}|))$ I/Os. Next compute a maximal matching of H_0, which takes $O(\mathrm{sort}(|H_0|))$ I/Os [17], and contract all edges in the matching. We call a vertex v of the resulting graph H'_0 *matched* if it is the result of contracting an edge in the matching. Otherwise, we call v *unmatched*. Our goal is to construct a graph H''_0 such that no unmatched vertex has an incident edge which is contractible.

Compute the subgraph $\tilde{H}$ of H'_0 induced by all edges incident to unmatched vertices. Graph $\tilde{H}$ is bipartite. Let V_u be the set of unmatched vertices and V_m be the set of matched vertices in $\tilde{H}$. Number the vertices in V_u and V_m in their order of appearance. Every vertex $v \in V_m$ stores the set of vertices in V_u adjacent to v, sorted by increasing numbers. Let w be a vertex adjacent to v, $v_1 < v_2 < \cdots < v_k$ be the vertices adjacent to w, and $v = v_j$. Then the copy of w in the adjacency list of v stores the number of vertex v_{j+1}. If v_{j+1} does not exist, v is being marked as being the last vertex adjacent to w. Also, v is being marked as the first vertex adjacent to w if $v = v_1$. This representation of $\tilde{H}$ can be constructed in $O(\mathrm{sort}(|H_0|)$ I/Os from the edge list of H'_0.

Now inspect the vertices in V_m in their order of appearance and use a priority queue Q to keep track of candidate edges for contraction. The edges in Q are directed and sorted lexicographically. For every vertex $v \in V_m$, let $w_1 < w_2 < \cdots < w_l$ be the vertices in V_u adjacent to v. For every vertex w_j do the following: If v is not the first vertex adjacent to w_j and the minimum edge in Q is not (v, w_j), proceed to the next vertex w_{j+1}. Otherwise, check whether $\omega(v) + \omega(w_j) \leq \rho_i$ and $\sigma(v) + \sigma(w_j) \leq 56$. If this is the case, contract edge $\{v, w_j\}$. Otherwise, there are two possibilities. If v is the last vertex adjacent to w_j, none of the edges incident to w_j is contractible. If v is not the last vertex, let u be the vertex stored with w_j in the adjacency list of v. Then add edge (u, w_j) to Q.

This procedure contracts an edge unless it is not contractible or its unmatched endpoint has already been contracted into another matched vertex, so that the edge now joins two matched vertices. Thus, none of the remaining edges that are incident to unmatched vertices are contractible. The I/O-complexity of this procedure is $O(\mathrm{sort}(|H_0|))$ [4], as $\tilde{H}$ is planar and its size is bounded from above by the size of H_0.

Let H''_0 be the graph obtained from H'_0 by contracting the edges in $\tilde{H}$ using the above algorithm. Extract the contractible subgraph H_1 of H''_0 and repeat the whole process until the contractible subgraph H_{s+1} of graph H''_s is empty. From the above discussion it follows that each iteration takes $O(\mathrm{sort}(|H_j|))$ I/Os. Moreover, the sizes of graphs $H_0, \ldots, H_s$ are geometrically decreasing, as every vertex in H_{j+1} is the result of contracting at least one edge in H_j. Thus, the whole contraction procedure takes $O(\mathrm{sort}(|G_{i-1}|))$ I/Os.

Once G_{i-1} has been compressed in this manner, it remains to partition the light vertices of degree at most two in the resulting graph G''_i into classes $C_1, \ldots, C_q$; each of these classes has to be partitioned into subclasses $C_{j,l}$ as described above. This takes sorting and scanning. Thus, G_i can be constructed from G_{i-1} in $O(\mathrm{sort}(|G_{i-1}|))$ I/Os. As $|G_0| = N$, and the sizes of subgraphs G_i are geometrically decreasing, we obtain the following lemma.

LEMMA 2.2. *The sequence $G = G_0, \ldots, G_r$ of graphs, as defined in Section 2.2, can be constructed in $O(\mathrm{sort}(N))$ I/Os using $O(N/B)$ blocks of external memory, where $N = |G|$.*

2.4 The Separator Hierarchy Having constructed graphs $G_0, \ldots, G_r$, we use them to construct a separator S of size $O\big(N/\sqrt{h}\big)$ whose removal partitions G into connected subgraphs of size at most $h \log^2(DB)$.

Using the planar embedding algorithm of [11] and Theorem 1.1, compute a partition of G_r into subgraphs of size at most $h \log^2(DB)$. As $|G_r| = O(N/(DB))$, this takes $O(N/(DB))$ I/Os. Let $S_r = S''_r$ be the computed separator, whose size is $|S_r| \leq c|G_r|/\big(\sqrt{h}\log(DB)\big)$, for some constant c defined in [3]. Given a separator S_{j+1} for graph G_{j+1}, compute a separator S_j for graph G_j as

follows: First construct the set S'_j of vertices represented by the vertices in S_{j+1}. S'_j is a separator of G_j whose removal partitions G_j into subgraphs of size at most $56h\log^2(DB)$. Load every subgraph H of $G_j - S_j$ whose size exceeds $h\log^2(DB)$ into internal memory, compute an embedding of H, and partition it into subgraphs of size at most $h\log^2(DB)$, again applying Theorem 1.1. Let S''_j be the separator obtained by partitioning all heavy subgraphs of $G_j - S'_j$ in this manner; $|S''_j| \leq c|G_j|/(\sqrt{h}\log(DB))$, for the same constant c as defined above. Now $S_j = S'_j \cup S''_j$. Iterate this procedure to compute a separator S_0 of $G_0 = G$.

LEMMA 2.3. *The separator S_0 of G computed by the above procedure has size $O(N/\sqrt{h})$. The connected components of $G - S_0$ have size at most $h\log^2(DB)$.*

Proof. It follows from the above construction and Lemma 2.1 that $|S_0| \leq \sum_{i=0}^{r} \rho_i |S''_i| \leq c\sum_{i=0}^{r} \rho_i \frac{|G_i|}{\sqrt{h}\log(DB)} \leq c\sum_{i=0}^{r} \rho_i \frac{28N}{\rho_i\sqrt{h}\log(DB)} = c\sum_{i=0}^{r} \frac{28N}{\sqrt{h}\log(DB)} = \frac{28cN}{\sqrt{h}}$. The bound on the size of the connected components of $G - S_0$ is explicitly ensured by our construction. □

Given separator S_{i+1}, it takes $O(\mathrm{sort}(|G_i|))$ I/Os to construct S'_i and compute the connected components of $G_i - S'_i$ [8]. Then it takes $O(\mathrm{scan}(|G_i|))$ I/Os to load each connected component into internal memory and perform the above partition. As the sizes of graphs $G_0, \ldots, G_r$ are geometrically decreasing, the total I/O-complexity of this part of the algorithm is $O(\mathrm{sort}(N))$.

2.5 Computing the Final Separator In order to obtain the final separator, each connected component of $G - S_0$ has to be partitioned into subgraphs of size at most h and boundary size at most $\sqrt{h}$.

First use Theorem 1.1 to partition the connected components of $G - S_0$ into subgraphs of size at most h. This takes $O(\mathrm{scan}(N))$ I/Os, as each component fits into internal memory. Let S' be the set of separator vertices introduced by partitioning the connected components of $G - S_0$ in this manner. By Theorem 1.1, $|S'| = O(N/\sqrt{h})$. Let $S'_0 = S_0 \cup S'$. Then $|S'_0| = O(N/\sqrt{h})$.

Next apply Theorem 1.2 to compute a separator $S \supseteq S'_0$ whose removal partitions G into $O(N/h)$ subgraphs whose boundary sizes do not exceed $\sqrt{h}$. In order to do this, we need to obtain a partition of $G - S'_0$ into $O(N/h)$ subgraphs $\bar{H}_1, \ldots, \bar{H}_s$ such that $|\bar{H}_i| \leq h$, for $1 \leq i \leq s$, and $\sum_{i=1}^{s} |\partial\bar{H}_i| = O(N/\sqrt{h})$. For the partition produced by our algorithm so far, the total boundary size of the connected components of $G - S'_0$ may be $\omega(N/\sqrt{h})$, even though $|S'_0| = O(N/\sqrt{h})$, and there may be as many as $\Omega(N)$ connected components of $G - S'_0$. Next we show how to meet the requirements of Theorem 1.2.

In order to reduce the total boundary size to $O(N/\sqrt{h})$, build a bipartite planar graph $\tilde{G}$ containing all separator vertices as well as one "region" vertex per connected component of $G - S'_0$. There is an edge between a region vertex w_R corresponding to some connected component R of $G - S'_0$ and a separator vertex v if $v \in \partial R$. The weight of a region vertex w_R is $\omega(w_R) = |R|$. As long as there are two region vertices v and w of degree at most two with the same set of neighbors and such that $\omega(v) + \omega(w) \leq h$, merge these vertices into a single vertex, and the corresponding subgraphs into a single subgraph. It follows from arguments similar to those in Section 2.2 that the resulting graph G' has $O(N/\sqrt{h})$ vertices. The total boundary size of all subgraphs is the same as the number of edges in G'. As G' is planar, this is $O(N/\sqrt{h})$. Assigning $\omega(v) = 0$ to all separator vertices $v \in G'$, the compression procedure of Section 2.2 can now be used to further merge subgraphs of $G - S'_0$, so that we finally obtain $O(N/h)$ subgraphs of size at most h each and total boundary size $O(N/\sqrt{h})$.

Even though the partition meets the requirements of Theorem 1.2 now, we cannot apply Theorem 1.2 to graphs $\bar{H}_i \cup \partial\bar{H}_i$ directly because the construction so far gives no bound on the size of $\partial\bar{H}_i$ better than $O(N/\sqrt{h})$, so that graph $\bar{H}_i \cup \partial\bar{H}_i$ may not fit into internal memory.

To overcome this difficulty, we represent each subgraph $\bar{H}_i$ and its boundary by another graph $\tilde{H}_i$, which is obtained by compressing $\partial\bar{H}_i$ to a new set $\partial\tilde{H}_i$, whose size is at most $6h$. We first define $\tilde{H}_i$ and prove that its size is at most $7h$. Then we show how to derive the desired separator of $\bar{H}_i$ from a separator of $\tilde{H}_i$.

In order to compute $\tilde{H}_i$ from $\bar{H}_i$, consider the bipartite graph induced by edges $\{v, w\}$, $v \in \bar{H}_i, w \in \partial\bar{H}_i$. Partition the vertices in $\partial\bar{H}_i$ into a minimal number of classes so that the vertices in each class are adjacent to the same set of vertices in $\bar{H}_i$. Let the *degree* of such a class C be the number of vertices in $\bar{H}_i$ adjacent to the vertices in C. For each class C of degree at most two, replace the vertices in C by a single vertex v of weight $\omega(v) = |C|$. For all other vertices $v \in \partial\bar{H}_i$, $\omega(v) = 1$. For all vertices in $\bar{H}_i$, $\omega(v) = 0$. By Corollary 1.1, $|\tilde{H}_i| \leq 7|\bar{H}_i| \leq 7h$. Now load $\tilde{H}_i$ into internal memory and apply Theorem 1.2 to compute a partition of $\tilde{H}_i$ into subgraphs $\tilde{H}_{i,1}, \ldots, \tilde{H}_{i,k_i}$ of weight at most $\sqrt{h}/2$ each. By Theorem 1.2, the total size of the separators computed for all graphs $\tilde{H}_1, \ldots, \tilde{H}_s$ is $O(N/\sqrt{h})$, and each subgraph $\tilde{H}_{i,j}$ of $\tilde{H}_i$ is adjacent to at most $\sqrt{h}/2$ separator vertices. However, some of these vertices may have a large weight; that is, they correspond to many separator vertices in $\partial\bar{H}_i$. Thus,

even though the vertices of a subgraph $\tilde{H}_{i,j}$ are adjacent to at most $\sqrt{h}/2$ separator vertices in $\tilde{H}_i$, they may be adjacent to many separator vertices in $\bar{H}_i \cup \partial\bar{H}_i$. But every separator vertex v of large weight is adjacent to at most two vertices in $\bar{H}_i$. Thus, replacing every such vertex with the adjacent vertices in $\bar{H}_i$ results in a separator of at most twice the size of the separator computed for $\tilde{H}_i$; but every subgraph of $\bar{H}_i$ is adjacent to at most $\sqrt{h}$ separator vertices now.

Let S'' be the set of separator vertices introduced in this step, and let $S = S'_0 \cup S''$ be the final separator. By Theorem 1.2, $|S| = O(N/\sqrt{h})$, and the number of subgraphs $\bar{H}_{i,j}$ obtained by removing the vertices in S from G is $O(N/h)$.

THEOREM 2.1. *Given a planar graph G and an integer $h > 0$, it takes $O(\mathrm{sort}(N))$ I/Os and $O(N/B)$ blocks of external memory to compute a separator S of size $O(N/\sqrt{h})$ whose removal partitions G into $O(N/h)$ subgraphs of size at most h and boundary size at most $\sqrt{h}$, provided that $M \geq 56h\log^2(DB)$.*

Note that our separator algorithm can be used to compute a small separator for G as long as graphs $G_0, \ldots, G_r$ satisfy Conditions (ii)–(iv) of Lemma 2.1, graph G_r is planar and all subgraphs of graphs $G_0, \ldots, G_r$ loaded into internal memory are planar. In particular, we can augment our algorithm to check for violation of any of these conditions; if a non-planar subgraph is detected or a graph G_i has not reduced in size sufficiently compared to graph G_{i-1}, the algorithm aborts and reports that G is not planar. If no violation is detected, graph G may still be non-planar; but the algorithm is guaranteed to produce a small separator in $O(\mathrm{sort}(N))$ I/Os.

3 Planarity Testing and Planar Embedding

3.1 Outline In this section, we outline an algorithm to test whether a given graph is planar and to derive a planar embedding of the graph if the answer is affirmative. Our algorithm is unlike any previous algorithm, as it uses a separator of the given graph, in order to derive the embedding. The main idea is to partition the graph G into subgraphs $G_1, \ldots, G_k$ using a small graph separator. Then replace each graph G_i by a constraint graph C_i whose size is linear in the number of separator vertices in G_i. These constraint graphs have the property that the graph A obtained by joining them at their separator vertices is planar if and only if G is planar. A planar embedding of G can be obtained from an embedding of A by locally replacing the planar embedding of each graph C_i with a consistent planar embedding of G_i. As the size of each constraint graph C_i is linear in the number of separator vertices in G_i, the size of A is linear in the size of the separator. Thus, choosing the separator to be of size $O(N/(DB))$, the size of graph A is $O(N/(DB))$, so that a planar embedding of A can be obtained in $O(N/(DB))$ I/Os [11].

The construction of the constraint graph C_i of a graph G_i is based on a decomposition of G_i into its connected components, which are then partitioned into their biconnected components (*bicomps*); each biconnected component is partitioned into its triconnected components (*tricomps*). (See appendix or [10] for a definition of triconnected components and related concepts.) The constraint graph of a tricomp can be computed rather easily, as the planar embedding of a triconnected planar graph is unique [22]. The constraint graph of a bicomp is obtained by merging the constraint graphs of its tricomps. Then compute the constraint graph of a connected component by joining the constraint graphs of its bicomps at their cutpoints. The following are the main steps of our algorithm.

Input: A simple connected graph $G = (V, E)$.

1: **if** $|E| > 3|V| - 6$ **then**
2: Report that G is not planar. **stop**.
3: Apply Theorem 2.1 to compute a separator S of G of size $O(N/(DB))$ whose removal partitions G into $O(N/(DB))$ subgraphs $G_1, \ldots, G_k$ such that $|G_i| \leq (DB)^2$ and $|\partial G_i| \leq DB$, for $1 \leq i \leq k$.
4: **if** the previous step fails **then**
5: Report that G is not planar. **stop**.
6: Let graphs $\bar{G}_1, \ldots, \bar{G}_k$ be defined as $\bar{G}_i = G[V(G_i) \cup \partial G_i]$, and let $G'_1, \ldots, G'_l$ be the connected components of graphs $\bar{G}_1, \ldots, \bar{G}_k$.
7: **for** $j = 1, \ldots, l$ **do**
8: **if** G'_j is not planar **then**
9: Report that G is not planar. **stop**.
10: Compute the constraint graph C_j of G'_j.
11: Let $A = G[S] \cup C_1 \cup \cdots \cup C_l$.
12: **if** A is not planar **then**
13: Report that G is not planar. **stop**.
14: Compute a planar embedding $\hat{A}$ of A.
15: **for** $j = 1, \ldots, l$ **do**
16: Replace the planar embedding of C_j induced by $\hat{A}$ with a consistent planar embedding of G'_j.
17: Output the resulting embedding $\hat{G}$ of G.

In Lines 4–5 we make use of the remark at the end of Section 2. In particular, we try to compute a small separator for G, not knowing whether G is planar. If the separator algorithm aborts, G cannot be planar. Otherwise, a small separator is produced in $O(\mathrm{sort}(N))$ I/Os. Given that we can construct graph A so that graph G is planar if and only if A is planar, and a planar embedding of G can be derived

from a planar embedding of A using local replacements, the correctness of the algorithm is now obvious. By Theorem 2.1, Step 3 of the algorithm takes $O(\text{sort}(N))$ I/Os, provided that $M \geq (DB)^2 \log^2(DB)$. Step 14 takes $O(N/(DB))$ I/Os, since we will show that $|A| = O(N/(DB))$. Steps 8–10 and 16 can each be carried out in internal memory, as each graph $\bar{G}_i$ has size at most $(DB)^2 + DB$, and thus each graph G'_j or C_j fits into internal memory. The iterative local replacement of graphs $C_1, \ldots, C_l$ by graphs $G'_1, \ldots, G'_l$ in Lines 15–16 requires some coordination between the different local replacement steps; this can be achieved in $O(\text{sort}(N))$ I/Os using time-forward processing. Details appear in the full paper [23]. The rest of this section describes the computation of constraint graphs $C_1, \ldots, C_l$, assuming that graphs $G'_1, \ldots, G'_l$ are planar.

3.2 The Constraint Graphs Given a graph G'_i, let $\mathcal{B}_1, \ldots, \mathcal{B}_q$ be its biconnected components. The bicomp-cutpoint-tree T_2 of G'_i contains one vertex β_r per bicomp $\mathcal{B}_r$ as well as all cutpoints of G'_i. There is an edge $\{v, \beta_r\}$ in T_2 if cutpoint v is contained in bicomp $\mathcal{B}_r$. Let S_i be the set of separator vertices in G'_i. For each vertex $v \in S_i$, let $\mathcal{B}(v)$ be a bicomp containing v. These bicomps are called *essential*. A bicomp $\mathcal{B}_r$ is called *potentially essential* if there are two essential bicomps $\mathcal{B}_s$ and $\mathcal{B}_t$ such that vertex β_r lies on the path from β_s to β_t in T_2. All other bicomps are called *inessential*. Let K be a connected component of the graph defined by all inessential bicomps of G'_i. Observe that K shares only a single vertex with the rest of G, as it does not contain any separator vertices, except possibly that cutpoint. Thus, after embedding the graph $G^{(1)}$ obtained by removing all inessential bicomps from G, these components K can easily be added to the embedding of $G^{(1)}$. In particular, G is planar if and only if $G^{(1)}$ is planar. Let $T_2^{(1)}$ be the tree obtained from T_2 by removing all subtrees of T_2 corresponding to inessential bicomps. Every leaf of $T_2^{(1)}$ corresponds to an essential bicomp of G'_i. A vertex of $T_2^{(1)}$ is *important* if it corresponds to an essential bicomp or has degree at least three. All other vertices of $T_2^{(1)}$ are *unimportant*. A bicomp is important if and only if its corresponding vertex in $T_2^{(1)}$ is important.

Partition the set of unimportant bicomps of G'_i into subsets so that each such subset corresponds to a maximal path of unimportant vertices in $T_2^{(1)}$. Let G_P be the graph obtained by merging all bicomps in such a subset. Graph G_P shares two cutpoints v and w with the rest of G. If G_P has a planar embedding so that v and w are embedded on the boundary of a common face, replace G_P by a single edge $\{v, w\}$.

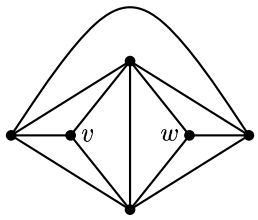

Figure 1: The constraint graph of graph G_P if vertices v and w cannot be on the same face.

Otherwise, replace G_P by a triconnected planar graph of constant size which contains v and w and does not allow v and w to be embedded on the boundary of a common face (Figure 1). In the former case, edge $\{v, w\}$ can be replaced by an embedding of G_P with v and w on the outer face and vice versa without affecting the embedding of the rest of G. In the latter case, if G is planar, vertices v and w are cutpoints in G, and any connected component of $G \setminus G_P$ can be adjacent to at most one of v or w. Thus, again, the embeddings of G_P and its constraint graph are interchangeable. Let $G^{(2)}$ be the graph obtained from $G^{(1)}$ after performing all these replacements. Then $G^{(2)}$ is planar if and only if $G^{(1)}$ is planar.

Next every important bicomp $\mathcal{B}$ of G'_i has to be replaced with its constraint graph $C_{\mathcal{B}}$. Let $R(\mathcal{B})$ be the set of *required* vertices of $\mathcal{B}$ defined as the set of all separator vertices and cutpoints in $\mathcal{B}$. Intuitively, these are the vertices that need to be present in $C_{\mathcal{B}}$, in order to join $C_{\mathcal{B}}$ with $G \setminus \mathcal{B}$. In order to compute $C_{\mathcal{B}}$, apply the same strategy as just described to the tricomps of $\mathcal{B}$. In particular, let $\mathcal{T}_1, \ldots, \mathcal{T}_q$ be the tricomps of $\mathcal{B}$. Then the recursive definition of these tricomps immediately gives rise to a tree structure, which we call the *tricomp tree* T_3 of $\mathcal{B}$. This tree contains a vertex τ_i for each tricomp $\mathcal{T}_i$. Two vertices τ_i and τ_j are adjacent if and only if tricomps $\mathcal{T}_i$ and $\mathcal{T}_j$ share a *virtual edge* (see appendix or [10]). For each required vertex $v \in R(\mathcal{B})$, let $\mathcal{T}(v)$ be a tricomp that contains v. We call these tricomps essential. Potentially essential and inessential tricomps are defined in the same way as potentially essential and inessential bicomps.

First remove all inessential tricomps. For every virtual edge (v, w, i) shared by an inessential and an essential or potentially essential tricomp $\mathcal{T}$, replace edge (v, w, i) by a "real" edge (v, w, i) in $\mathcal{T}$. Let $G^{(3)}$ be the graph obtained from $G^{(2)}$ using this operation. Consider the subgraphs of $G^{(2)}$ obtained by merging adjacent inessential tricomps. Each such subgraph K contains one virtual edge (v, w, i), which it shares with an essential or potentially essential tricomp. Edge (v, w, i) in $G^{(3)}$ can be replaced by a planar embedding

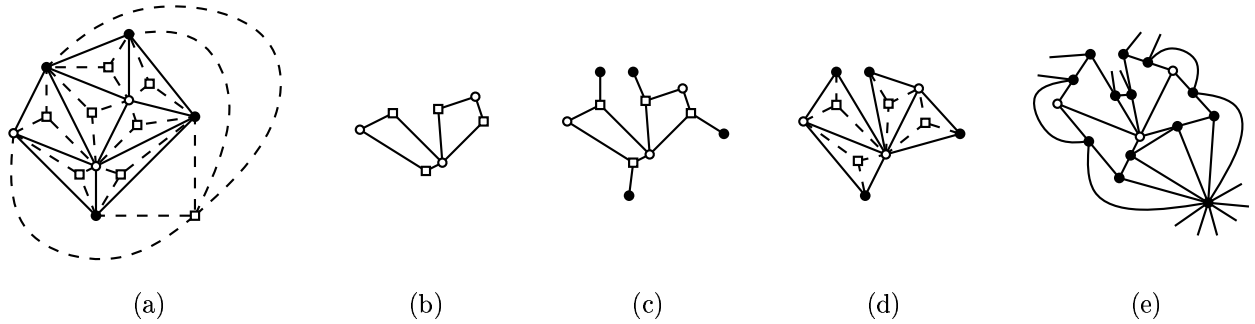

Figure 2: (a) A tricomp $\mathcal{T}$ (solid edges) and its face-on-vertex graph G_F (dashed edges). Required vertices are white disks. (b) Graph G'_F (c) Graph G''_F. (d) Graph $C'_\mathcal{T}$. (e) The constraint graph $C_\mathcal{T}$ of $\mathcal{T}$.

of K without affecting the planar embedding of the rest of the graph. Thus, $G^{(3)}$ is planar if and only if $G^{(2)}$ is planar. Let $T_3^{(3)}$ be the tree obtained from T_3 by removing all vertices in T_3 corresponding to inessential tricomps. Then every leaf in $T_3^{(3)}$ corresponds to an essential tricomp. Again, a vertex of $T_3^{(3)}$ is important if it either corresponds to an essential tricomp, or it has degree at least three. The remaining vertices are unimportant. These vertices can be partitioned into maximal paths consisting of only unimportant vertices. Let P be such a path and G_P be the graph obtained by merging all tricomps corresponding to the vertices on P. Then G_P contains two virtual edges (a, b, i) and (c, d, j). Replace graph G_P by a constant-size constraint graph C_P representing the possible embeddings of edges (a, b, i), (c, d, j), and vertices a, b, c, and d with respect to each other. (See [23] for details.)

The final step is the replacement of each important tricomp $\mathcal{T}$ with a constraint graph $C_\mathcal{T}$. Let $R(\mathcal{T})$ be the set of required vertices in $\mathcal{T}$. A vertex is required in $\mathcal{T}$ if it is either part of a *separation pair* (see appendix or [10]) or in $R(\mathcal{B})$. If $\mathcal{T}$ is a *bond* (see appendix or [10]), $C_\mathcal{T} = \mathcal{T}$. If $\mathcal{T}$ is a simple cycle, $C_\mathcal{T}$ is the graph obtained by iteratively contracting edges incident to vertices that are not required until only required vertices remain. In order to construct $C_\mathcal{T}$ for a simple triconnected graph $\mathcal{T}$, compute the face-on-vertex graph G_F of the unique planar embedding $\hat{\mathcal{T}}$ of $\mathcal{T}$ (Figure 2a). G_F contains all vertices of $\mathcal{T}$ as well as a vertex w_f for each face f of $\hat{\mathcal{T}}$. There is an edge between a vertex v of $\mathcal{T}$ and a face vertex w_f if v appears on the boundary of face f. The embedding $\hat{\mathcal{T}}$ defines a planar embedding $\hat{G}_F$ of G_F in a natural manner. Remove all vertices $v \in V(\mathcal{T}) \setminus R(\mathcal{T})$ from G_F. Also remove all face vertices that are isolated in the resulting graph. Let G'_F be the resulting graph containing all required vertices of $\mathcal{T}$ as well as all face vertices corresponding to faces with at least one required vertex on their boundaries. For every vertex $v \in R(\mathcal{T})$, remove as many degree-1 vertices adjacent to v as possible without decreasing the degree of v below two (Figure 2b). For every face vertex of degree less than three, add one or two dummy vertices adjacent to this face vertex, in order to increase its degree to three (Figure 2c). This is necessary so that the resulting graph G''_F can be made a subgraph of the face-on-vertex graph of the constraint graph $C_\mathcal{T}$. In order to construct $C_\mathcal{T}$, connect the required and dummy vertices clockwise around each face vertex. Let $C'_\mathcal{T}$ be the graph obtained by removing all face vertices from the resulting graph, and $\hat{C}'_\mathcal{T}$ be the embedding of $C'_\mathcal{T}$ induced by the embedding of G''_F (Figure 2d). Most faces in $C'_\mathcal{T}$ correspond to face-vertices in G''_F. Partition faces that do not correspond to such a face vertex in a manner that makes the resulting graph triconnected and guarantees that every face not corresponding to a vertex in G''_F has at most one required vertex on its boundary. The resulting constraint graph $C_\mathcal{T}$ (Figure 2e) has the property that the faces with two or more required vertices on their boundaries in $\hat{\mathcal{T}}$ and $\hat{C}_\mathcal{T}$ are in one-to-one correspondence so that two corresponding faces $f \in \hat{\mathcal{T}}$ and $f' \in \hat{C}_\mathcal{T}$ have the same required vertices on their boundaries, in the same order. As a result, graphs $\mathcal{T}$ and $C_\mathcal{T}$ are interchangeable in any planar embedding of G, as every subgraph of G embedded in face f of $\hat{\mathcal{T}}$ can be embedded without modifications in the corresponding face f' of $\hat{C}_\mathcal{T}$ and vice versa. Let A be the graph obtained from $G^{(3)}$ by replacing all important tricomps with their constraint graphs. Then graph A is planar if and only if $G^{(3)}$ is planar. This implies that A is planar if and only if G is planar.

It is shown in [23] that the constraint graph $C_\mathcal{T}$ of an important tricomp $\mathcal{T}$ has size $O(|R(\mathcal{T})|)$. For an important bicomp $\mathcal{B}$, let $T_3^{(4)}$ be the tree obtained from $T_3^{(3)}$ by compressing each path whose internal vertices are unimportant to a single edge. Then all vertices in $T_3^{(4)}$ correspond to important tricomps. Every edge of

$T_3^{(4)}$ is represented by a constant size constraint graph in $C_{\mathcal{B}}$. $T_3^{(4)}$ is easily shown to have size $O(|R(\mathcal{B})|)$, so that the total size of the constraint graphs corresponding to edges in $T_3^{(4)}$ is $O(|R(\mathcal{B})|)$. Also, the total number of required vertices in all essential tricomps of $\mathcal{B}$ is $O(|R(\mathcal{B})|)$, so that the total size of all constraint graphs $C_{\mathcal{T}}$ in $C_{\mathcal{B}}$ is $O(|R(\mathcal{B})|)$. Thus, $|C_{\mathcal{B}}| = O(|R(\mathcal{B})|)$. A similar argument applied to the bicomps and the bicomp-cutpoint-tree of G'_i shows that the constraint graph C_i of G'_i has size $O(|S_i|)$. This implies that $|A| = |G[S]| + \sum_{i=1}^{l} |C_i| = O(N/(DB)) + \sum_{i=1}^{l} |S_i| = O(N/(DB))$, so that we obtain the following result.

THEOREM 3.1. *Provided that $M \geq (DB)^2 \log^2(DB)$, it takes $O(\mathrm{sort}(|V|))$ I/Os and $O((|V| + |E|)/B)$ blocks of external memory to decide whether a given graph $G = (V, E)$ is planar and to compute a planar embedding of G if the answer is affirmative.*

4 Applications

In this section, we apply Theorems 2.1 and 3.1 to solve BFS, DFS, and the single source shortest path problem on planar graphs with non-negative edge weights as well as compute a weighted ϵ-separator I/O-efficiently. Given that SSSP can be solved in $O(\mathrm{sort}(N))$ I/Os, the results on BFS and DFS follow from [6] and the fact that BFS is SSSP in a graph with unit edge weights. To solve SSSP, compute a separator of size $O(N/(DB))$ partitioning G into $O(N/(DB)^2)$ subgraphs of size at most $(DB)^2$ and boundary size at most DB, and apply the algorithm of [5]. The algorithm of [5] is I/O-efficient only if the input graph has bounded degree. To satisfy this constraint while maintaining planarity, replace every vertex v of degree greater than three by a cycle and connect each edge incident to v to a unique vertex in the cycle, maintaining the order of the edges around v. This construction can easily be carried out in $O(\mathrm{sort}(N))$ I/Os. The resulting graph G' has size $O(N)$. Assigning weight 0 to each edge in the cycle, it is easy to show that the distance between two vertices in G is the same as the distance between any two vertices in the corresponding cycles in G'.

Another requirement that has to be satisfied in order to apply the algorithm of [5] is that the number of boundary sets produced by the separator algorithm is small. The concept of boundary sets was introduced in [9]. A *boundary set* is a maximal set of separator vertices adjacent to the same set of subgraphs of $G' \setminus S$. The algorithm of [5] requires that there be only $O(N/(DB)^2)$ boundary sets. In order to ensure this requirement, partition $G' \setminus S$ into its connected components $Q_1, \ldots, Q_r$ and construct a graph $\tilde{G}$ containing one vertex v_i per connected component Q_i. There is an edge between two vertices v_i and v_j if Q_i and Q_j are adjacent to a common separator vertex. As the vertices in G' have degree at most three, $\tilde{G}$ is planar. Assign two weights $\omega(v_i) = |Q_i|$ and $\gamma(v_i) = |\partial Q_i|$ to each vertex v_i. Then apply the compression procedure of Section 2.2 to $\tilde{G}$ so that every vertex v in the resulting graph satisfies $\omega(v) \leq (DB)^2$ and $\gamma(v) \leq DB$. This compression corresponds to merging the connected components of $G' \setminus S$ into $O(N/(DB)^2)$ subgraphs $G_1, \ldots, G_l$ of size at most $(DB)^2$ and boundary size at most DB. Moreover, it is easily verified that each graph $R_i = G_i \cup \partial G_i$ is either connected or intersects with at most two other graphs R_j and R_k, which are connected. Now it follows from [9] that there are only $O(N/(DB)^2)$ boundary sets. We summarize this section in the following theorem.

THEOREM 4.1. *It takes $O(\mathrm{sort}(N))$ I/Os and $O(N/B)$ blocks of external memory to solve BFS, DFS, SSSP on a planar graph G of size N whose edges have non-negative weights, provided that $M \geq (DB)^2 \log^2(DB)$.*

Given a BFS-tree and an embedding of a planar graph G, the algorithm of [3] can be made to take only $O(\mathrm{sort}(N))$ I/Os to compute a weighted separator of a planar graph G. (Details appear in the full paper.)

THEOREM 4.2. *Given a planar graph $G = (V, E)$ whose vertices have non-negative weights $\omega(v)$ so that $\sum_{v \in G} \omega(v) \leq 1$, and a constant $0 < \epsilon < 1$, it takes $O(\mathrm{sort}(N))$ I/Os and $O(N/B)$ blocks of external memory to compute a separator S of size $O(\sqrt{N/\epsilon})$ so that no connected component of $G - S$ has weight exceeding ϵ, provided that $M \geq (DB)^2 \log^2(DB)$.*

5 Conclusions

We have presented I/O-efficient algorithms for a number of fundamental problems on planar graphs. These algorithms take $O(\mathrm{sort}(N))$ I/Os, provided that $M \geq (DB)^2 \log^2(DB)$. Even though this requirement seems unreasonable, our separator algorithm requires less internal memory than that of [5]; but the algorithm of [5] also requires a BFS-tree and an embedding to be part of the input, while our algorithm can do without them.

Using Theorem 4.2, we can apply a bootstrapping approach to reduce the amount of memory required by our algorithm, provided that the SSSP problem can be solved in $O(\mathrm{sort}(|E|))$ I/Os under the assumption that the vertices fit into internal memory (the *semi-external* scenario). Only a constant number of bootstrapping rounds would be required to reduce the memory requirement to the smallest reasonable bound of $M \geq DB$. Thus, the complexity of our algorithm would increase by only a constant factor. Given this observation, we consider finding an optimal semi-external SSSP algorithm to be a very important open problem.

Acknowledgements We would like to thank Lyudmil Aleksandrov for many helpful discussions.

References

[1] J. Abello, A. L. Buchsbaum, J. Westbrook. A functional approach to external graph algorithms. *Proc. ESA*, pp. 332–343, 1998.

[2] P. K. Agarwal, L. Arge, T. M. Murali, K. R. Varadarajan, J. S. Vitter. I/O-efficient algorithms for contour-line extraction and planar graph blocking. *Proc. SODA*, pp. 117–126, 1998.

[3] L. Aleksandrov, H. Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM J. Disc. Math.*, 9:129–150, 1996.

[4] L. Arge. The buffer tree: A new technique for optimal I/O-algorithms. *Proc. WADS*, pp. 334–345, 1995.

[5] L. Arge, G. S. Brodal, L. Toma. On external memory MST, SSSP, and multi-way planar separators. *Proc. SWAT*, pp. 433–447, 2000.

[6] L. Arge, U. Meyer, L. Toma, N. Zeh. On external-memory planar depth first search. *Proc. WADS*, pp. 471–482, 2001.

[7] J. Boyer, W. Myrvold. Stop minding your P's and Q's: A simplified $O(n)$ planar embedding algorithm. *Proc. SODA*, pp. 140–146, 1999.

[8] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, J. S. Vitter. External-memory graph algorithms. *Proc. SODA*, pp. 139–149, 1995.

[9] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comp.*, 16:1004–1022, 1987.

[10] J. Hopcroft, R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comp.*, 2:135–158, 1973.

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

[12] D. Hutchinson, A. Maheshwari, N. Zeh. An external memory data structure for shortest path queries. *Proc. COCOON*, pp. 51–60, 1999. To appear in Disc. Appl. Math.

[13] P. Klein, S. Rao, M. Rauch, and S. Subramanian. Faster shortest path algorithms for planar graphs. *J. Comp. Sys. Sci.*, 55:3–23, 1997.

[14] V. Kumar, E. J. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. *Proc. SPDP*, pp. 169–176, 1996.

[15] R. J. Lipton, R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36:177–189, 1979.

[16] A. Maheshwari, N. Zeh. External memory algorithms for outerplanar graphs. *Proc. ISAAC*, pp. 307–316, 1999.

[17] A. Maheshwari, N. Zeh. I/O-efficient algorithms for graphs of bounded treewidth. *Proc. SODA*, pp. 89–90, 2001.

[18] U. Meyer. External memory BFS on undirected graphs with bounded degree. *Proc. SODA*, pp. 87–88, 2001.

[19] K. Munagala, A. Ranade. I/O-complexity of graph algorithms. *Proc. SODA*, pp. 687–694, 1999.

[20] J. S. Vitter. External memory algorithms and data structures. In J. Abello and J. S. Vitter, eds., *External Memory Algorithms and Visualization*. AMS, 1999.

[21] J. S. Vitter, E. A. M. Shriver. Algorithms for parallel memory I: Two-level memories. *Algorithmica*, 12:110–147, 1994.

[22] H. Whitney. Congruent graphs and the connectivity of graphs. *Amer. J. Math.*, 54:150–168, 1932.

[23] N. Zeh. I/O-efficient planar embedding using graph separators. TR-01-07, School of Comp. Sci., Carleton University, 2001.

[24] N. Zeh. I/O-efficient planar separators and applications. TR-01-02, School of Comp. Sci., Carleton University, 2001.

Appendix: Triconnected Components

In this appendix, we define the most important concepts relating to the triconnected components of a biconnected graph G. Given a subgraph H of G, the bridges of H in G are defined as follows: Let the vertices in $V(G) \setminus V(H)$ be partitioned into equivalence classes such that for any two vertices v and w in such a class K, there exists a path from v to w in G which does not contain a vertex in $V(H)$. Each such class K gives rise to a *non-trivial bridge* B of H defined as the graph induced by all edges in G incident to the vertices in K. A *trivial bridge* is an edge $\{v, w\} \notin H$ such that $v, w \in V(H)$. The trivial and non-trivial bridges of H are its *bridges*.

A pair $\{v, w\}$ of vertices is a *separation pair* if the graph $H = (\{v, w\}, \emptyset)$ has either at least two non-trivial bridges or at least three bridges one of which is non-trivial. Given a separation pair $\{v, w\}$ with bridges $B_1, \ldots, B_q$, a *split* $s(v, w, i)$ chooses two graphs $B' = B_1 \cup \cdots \cup B_{q'}$ and $B'' = B_{q'+1} \cup \cdots \cup B_q$ such that $|E(B')| \geq 2$ and $|E(B'')| \geq 2$, and partitions G into two subgraphs $G_1 = (V(B'), E(B') \cup (v, w, i))$ and $G_2 = (V(B''), E(B'') \cup (v, w, i))$. Edge (v, w, i) is called the *virtual edge* corresponding to split $s(v, w, i)$. The *split components* of G are defined as the graphs obtained by recursively splitting G_1 and G_2 until there are no more separation pairs. The split components of G are not necessarily unique. There are three types of split components: (1) triconnected simple graphs, (2) triple bonds (two vertices with three edges between them), and (3) triangles. Now merge all triple bonds with the same two endpoints into a single bond, and merge all triangles sharing virtual edges. The resulting graphs are the *triconnected components* of G. The triconnected components of G are unique and of three types: (1) triconnected simple graphs, (2) bonds, and (3) simple cycles.