Simplified External Memory Algorithms for Planar DAGs

Lars Arge[1,*] and Laura Toma[2]

[1] Duke University, Durham, NC 27708, USA, E-mail: large@cs.duke.edu
[2] Bowdoin College, Brunswick, ME 04011, USA, E-mail: ltoma@bowdoin.edu

Abstract. In recent years a large number I/O-efficient algorithms have been developed for fundamental planar graph problems. Most of these algorithms rely on the existence of small planar separators as well as an $O(\mathrm{sort}(N))$ I/O algorithm for computing a partition of a planar graph based on such separators, where $O(\mathrm{sort}(N))$ is the number of I/Os needed to sort N elements.
In this paper we simplify and unify several of the known planar graph results by developing linear I/O algorithms for the fundamental single-source shortest path, breadth-first search and topological sorting problems on *planar directed acyclic graphs*, provided that a partition is given; thus our results give $O(\mathrm{sort}(N))$ I/Os algorithms for the three problems. While algorithms for all these problems were already known, the previous algorithms are all considerably more complicated than our algorithms and use $\Theta(\mathrm{sort}(N))$ I/Os even if a partition is known. Unlike the previous algorithm, our topological sorting algorithm is simple enough to be of practical interest.

1 Introduction

Recently, external memory graph algorithms have received considerable attention because massive graphs arise naturally in a number of applications such as transportation networks and geographic information systems (GIS). When working with massive graphs, the I/O-communication, and not the internal memory computation, is often the bottleneck. Efficient external-memory (or I/O-efficient) algorithms can thus lead to considerable runtime improvements.

The need for solving fundamental graph problems (such as topological sorting) on *planar* graphs often appear in e.g. GIS. For example, in an application such as flow modeling on grid terrain models, each cell in the terrain model is assigned a flow direction to one of its neighbors such that the resulting graph is planar and acyclic. To trace the amount of flow through each cell of the terrain one then needs to topologically sort this graph [5]; external memory algorithms are needed to do so efficiently, as modern terrain models—and thus the manipulated planar graphs—are often massive since projects such as NASAs EOS [1]

* Supported in part by the National Science Foundation through RI grant EIA–9972879, CAREER grant CCR–9984099, ITR grant EIA–0112849, and U.S.–Germany Cooperative Research Program grant INT–0129182.

and Space Radar Topography Mission [16] have acquired terrabytes of terrain data in recent years.

Even though a large number of I/O-efficient graph algorithms have been developed, a number of fundamental problems on general graphs still remain open. For planar graphs, on the other hand, significant progress has been made. A large number of fundamental problems on *undirected* planar graphs have been solved I/O-efficiently [3, 4, 9, 14] and recently several fundamental problems have also been solved for *directed* planar graphs [6, 7]. Most of these algorithms are based on the existence of small planar separators.

In this paper we simplify and unify several of the directed planar graph results by developing linear I/O algorithms for the fundamental single-source shortest path, breadth-first search and topological sorting problems on planar *directed acyclic graphs*, *provided* that a partition is given. Our algorithms rely on a set of reductions using the partition, which exploit the acyclicity in important ways. Previous algorithms all use more than linear I/Os even if a separation is known; they are also considerably more complicated than our new algorithms.

1.1 Problem statement

Let $G = (V, E)$ be a directed acyclic graph (DAG). We say that G is planar if it can be embedded in the plane such that no edges intersect. The *topological sorting* problem is the problem of computing an order on the vertices of G so that for any edge $(u, v) \in E$, vertex u comes before vertex v in this order; a graph can be topologically sorted *if and only if* it is acyclic. If G is weighted, the *single-source shortest path* (SSSP) problem is the problem of finding the shortest paths from a given source vertex s in G to all other vertices in G, where the length of a path is defined as the sum of the weights of the edges on the path. The *breadth-first search* (BFS) problem is equivalent to SSSP where all edges have weight one.

1.2 I/O-model and previous results

We will be working in the standard two-level I/O model [2], where M is the number of vertices that can fit into internal memory, and B is the number of vertices that can fit into a disk block, with $M < N$ and $1 \leq B \leq \sqrt{M}$.[3] An *I/O* is the operation of transferring a block of data between main memory and disk, and the complexity of an algorithm is measured in terms of the number of disk blocks and I/Os it uses to solve a problem. The minimal number of I/Os needed to read N input elements (the "linear bound") is obviously $\text{scan}(N) = \Theta(N/B)$. The number of I/Os needed to sort N elements is $\text{sort}(N) = \Theta(\frac{N}{B} \log_{M/B} N/B)$ [2]. For all realistic values of N, B, and M, $\text{scan}(N) < \text{sort}(N) \ll N$, and the

[3] Some algorithms, like the planar separator algorithm of [14], make the stronger but realistic assumption that $M > B^2 \lg^2 B$. The algorithms described in this paper make this assumption indirectly as they rely on planar separators.

difference in running time between an algorithm performing N I/Os and one performing scan(N) or sort(N) I/Os can be very significant.

Despite considerable efforts, many fundamental problems on general graphs remain open; refer to the surveys in [15, 17] and the references therein. On general digraphs the best known algorithm for SSSP, as well the best algorithms for the simpler BFS and DFS traversal problems, use $O(\min\{(|V| + \frac{|E|}{B}) \cdot \log|V| + \text{sort}(|E|), |V| + \frac{|V|}{M}\frac{|E|}{B}\})$ I/Os [8, 9, 12] . Thus all these algorithms use $\Omega(V)$ I/Os, while the lower bound for the number of I/Os required to solve most graph problems is $\Omega(\min\{V, \text{sort}(V)\})$ (which, in practice is $\Omega(\text{sort}(V))$). As a result, improved algorithms have recently been developed for special classes of graphs. On *planar* digraphs SSSP, BFS, ear decomposition, as well as topological sorting of an acyclic graph have been solved in $O(\text{sort}(N))$ I/Os [6], while DFS can be solved in $O(\text{sort}(N)\log\frac{N}{M})$ I/Os [7]. All these algorithms are based on I/O-efficient reductions [3, 4, 6, 7] and on an $O(\text{sort}(N))$ I/O planar graph separator algorithm [14].

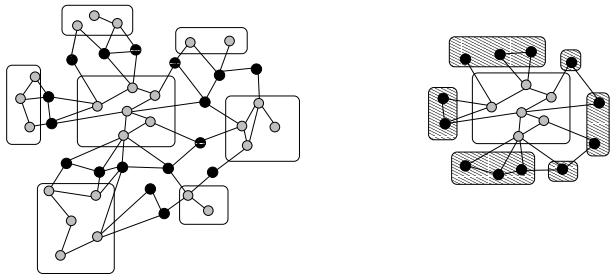

Fig. 1. (a) Partition of G into clusters G_i (boxed) and separators vertices V_S (black). (b) One cluster G_i in the partition and its adjacent boundary sets. For simplicity the direction of the edges is not shown.

Almost all of the above mentioned I/O-efficient algorithms for planar graphs utilize the existence of small separators. An $f(N)$-*separator* of an N-vertex graph $G = (V, E)$ is a subset V_S of the vertices V of size $f(N)$, such that the removal of V_S partitions G into two subgraphs G_1 and G_2 of size at most $\frac{2N}{3}$. Lipton and Tarjan [13] showed that any planar graph has an $O(\sqrt{N})$-separator. Using this result recursively, Frederickson [11] showed that for any parameter $R \in [1, N]$, there exists a subset V_S of $\Theta(N/\sqrt{R})$ vertices, such that the removal of V_S partitions G into $\Theta(N/R)$ subgraphs G_i of size $O(R)$, where (the vertices in) each G_i is (are) adjacent to $O(\sqrt{R})$ vertices of V_S. We call such a partitioning an R-*partition*. The vertices in V_S are called the *separator vertices* and each of the graphs G_i a *cluster*. The set of separator vertices adjacent to G_i are called the *boundary vertices* ∂G_i (or simply the *boundary*) of G_i. We use $\overline{G_i}$ to denote the graph consisting of G_i, ∂G_i and the subset of edges of E connecting G_i and

∂G_i. Refer to Fig. 1(a). The set of separator vertices can be partitioned into maximal subsets so that the vertices in each subset are adjacent to the same set of clusters. These sets are called the *boundary sets* of the partition. If the graph has bounded degree (which can be ensured for planar graphs using a simple transformation [11]), Frederickson showed that there exists an R-partition with only $O(N/R)$ boundary sets. Refer to Fig. 1(b). Maheshwari and Zeh showed how to compute such an R-partition in $O(\text{sort}(N))$ I/Os, provided that $M > B^2 \log^2 B$ [14].

1.3 Our results

In this paper we simplify and unify several of the known planar graph results by developing $O(\text{scan}(N))$ I/O algorithms for the fundamental single-source shortest path, breadth-first search and topological sorting problems on planar DAGs, *provided* that a B^2-partition is given. Since such a partition can be computed in $O(\text{sort}(N))$ I/Os, our results give new $O(\text{sort}(N))$ I/Os algorithms for the three problems. While such algorithms were already known, the previous algorithms are all *considerably* more complicated than our algorithms and use $\Theta(\text{sort}(N))$ I/Os even if a partition of the graph is known; however the previous BFS and SSSP algorithms work on general planar digraphs. Especially the previous topological sorting algorithm due to Arge, Toma, and Zeh [6], which utilizes SSSP and computation of a directed ear decomposition of a strongly connected directed planar graph, is much more complex than our algorithm; we show that given a B^2-partition topological sorting of an N vertex planar DAG can very easily be reduced in $O(\text{scan}(N))$ I/Os to topological sorting of a (non planar) DAG with $O(N/B)$ vertices and $O(N)$ edges, which in turn can easily be solved in $O(\text{scan}(N))$ I/Os using a slightly modified version of a simple internal memory algorithm—unlike the previous algorithm, our algorithm is simple enough to be of practical interest. Our results unify many of the previous results by showing that computing a good partition is the hard part of most planar DAG problems.

2 Topologically sorting a planar DAG

In this section we show how, given a B^2-partition of a planar DAG G with N vertices, we can topologically sort G in $O(\text{scan}(N))$ I/Os. Recall that a B^2-partition consist of $O(\frac{N}{B^2})$ clusters G_i of size $O(B^2) = O(M)$, each adjacent to $O(B)$ boundary vertices ∂G_i. We assume without loss of generality that G has bounded degree, and that the B^2-partition has $O(\frac{N}{B^2})$ boundary sets (sets of separator vertices adjacent to the same constant-sized set of clusters). Furthermore, we assume that G is given in edge-list representation, that is, as a list of edges with edges incident to each vertex (both incoming and outgoing) appearing consecutively in the list;[4] we assume that the edges incident to vertices in each cluster G_i are stored consecutively, and so are the edges incident to

[4] Any (reasonable) representation can be transformed into this representation in $O(\text{sort}(N))$ I/Os.

each boundary set. Note that this means that the graph $\overline{G_i}$ induced by G_i and ∂G_i can be loaded into main memory in $O(B)$ I/Os, and that the $O(B)$ edges incident to each boundary set can be loaded in $O(1)$ I/Os.

Our algorithm consists of a series of reductions, each of which can be performed in $O(\text{scan}(N))$ I/Os: First we reduce the problem of topologically sorting the DAG G to the problem of computing longest paths in a DAG G^s with $N+1$ vertices and $O(N)$ edges. We then show how to reduce this problem, using the B^2-partition of G, to computing longest paths in a weighted DAG G^R with $O(N/B)$ vertices and $O(N)$ edges. This problem can in turn be reduced to computing a topological sorting of the vertices in G^R, which we finally are able to solve efficiently directly because of the reduced number of vertices (as well as properties of the B^2-partition of G). Note that since computing longest paths is NP-complete on general graphs [10], it is somewhat surprising that we are able to topologically sort G by reducing the problem to computing longest paths. Note also that the first two and last two reductions can easily be combined, resulting in a relatively simple overall algorithm. Below we describe each of these steps (Lemmas 2, 6, 7, and 8) and thus prove the following:

Theorem 1. *Given a B^2-partition of a planar DAG G in edge-list representation, G can be topologically sorted G in $O(scan(N))$ I/Os.*

2.1 Reducing topological sorting of G to longest paths in G^s

Our first reduction simply consists of introducing a new source (indegree-zero) vertex s and adding edges to all indegree-zero vertices in G. We call the resulting graph G^s. Note that G^s is still a DAG but not necessarily planar. Now for each vertex v let $\lambda[v]$ be the length (number of edges) of the *longest* path from s to v. We can easily show that computing longest paths in G^s corresponds to topologically sorting G:

Lemma 1. *An ordering of the vertices in G^s by longest-path lengths $\lambda[v]$ is a topological ordering of the vertices in G.*

Proof. We must prove that $\lambda[u] < \lambda[v]$ for any $(u,v) \in E$. Assume by contradiction that $\lambda[u] \geq \lambda[v]$. Let p be the longest path from s to u of length $\lambda[u]$. Then the path p' obtained by adding the edge (u,v) to p is a path from s to v. Since this path has length $\lambda[u]+1$ we have $\lambda[v] \geq \lambda[u]+1$, which implies that $\lambda[u] < \lambda[v]$. □

Since we can add s to G and compute the $O(N)$ extra edges in a scan of G we have obtained the following:

Lemma 2. *Topologically sorting G can be reduced in $O(scan(N))$ I/Os to computing longest-path lengths from s to all vertices in G^s.*

2.2 Reducing longest paths in G^s to longest paths in G^R

In our second reduction we utilize the given B^2-partition of G, which we assume is stored implicitly in G^s. In fact, we assume that G^s is represented as the list of edges in G (ordered as discussed in the beginning of this section), and that the edges incident to s are represented implicitly by indegree-zero vertices, that is, we do not store s or its incident edges explicitly. Therefore, even though G^s is not planar, we will in the following refer to the B^2-partition of G^s.

The reduced graph G^R is defined as follows: The vertices of G^R consist of s and the separator vertices in the B^2-partition of G^s. To define the edges of G^R we first consider each cluster G_i in turn and, for every pair of vertices u and v on the boundary ∂G_i of G_i, we add an edge (u, v) if there is a path from u to v in $\overline{G_i}$ (that is, in the graph induced by $G_i \cup \partial G_i$); the edge (u, v) has weight equal to the length of the *longest* path from u to v in $\overline{G_i}$. In addition, for each vertex $u \in \overline{G_i}$ with an edge (s, u) in G^s (i.e. indegree-zero vertices in G) we also add edges (s, v) for all $v \in \partial G_i$ with a path from u to v in $\overline{G_i}$; the edge (s, v) has weight equal to one plus the length of the longest path from u to v in $\overline{G_i}$. Finally, we add all edges between two separator vertices in G^s; these edges have weight one.

G^R has $O(N/B)$ vertices since the number of separator vertices in the B^2-partition of G^s is $O(N/B)$. Since each of the $O(\frac{N}{B^2})$ clusters G_i has $O(B)$ boundary vertices ∂G_i, G^R has $O(B^2)$ edges between vertices in ∂G_i, as well as $O(B)$ edges between s and vertices in ∂G_i. Thus G^R has $O(N)$ edges in total.

In order to prove that the lengths (weights) of the longest paths to separator vertices are the same in G^s and G^R, we need the following general lemma:

Lemma 3. *Subpaths of longest paths in a DAG are longest paths.*

Proof. Let p be the longest path between two vertices u and v in a DAG G. Let u_1 and u_2 be two vertices on p and let $p_{u_1u_2}$ be the subpath of p between u_1 and u_2. By contradiction, assume that $p_{u_1u_2}$ is not the longest path from u_1 to u_2 in G, that is, there exists a path p' from u_1 to u_2 that is longer than $p_{u_1u_2}$. Since G is a DAG, p' cannot contain a vertex that appears on p before u_1 or after u_2 (if it contained such a vertex w, say, after u_2, there would be a cycle in G containing u_2 and w). Thus we can replace $p_{u_1u_2}$ in p by p' and obtain a path from u to v that is longer than p. This contradicts that p is the longest path from u to v. □

Lemma 4. *For each separator vertex v in G^R, the longest-path length $\lambda_R[v]$ between s and v in G^R is equal to the longest-path length $\lambda[v]$ in G^s.*

Proof. Consider the longest path p from s to a separator vertex v in G^s; let u_1 and u_2 be two consecutive separator vertices on p on the boundary of the same cluster G_i; that is, $u_1, u_2 \in \partial G_i$ and all vertices between u_1 and u_2 on p are in G_i; refer to Fig. 2(a). Since there is a path from u_1 to u_2 in G_i, there must be an edge (u_1, u_2) in G^R; this in particular means that there is a path from s to v in G^R. By definition, the weight of edge (u_1, u_2) in G^R is equal to the length of the

longest path from u_1 to u_2 in $\overline{G_i}$. By Lemma 3 the length of this path must be the same as the length of the subpath of p from u_1 to u_2. Thus $\lambda_R[v] = \lambda[v]$. □

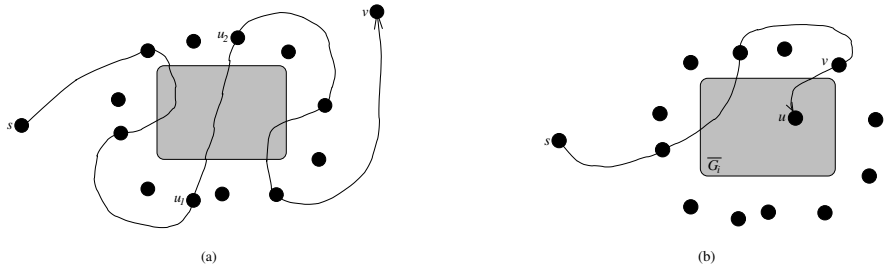

Fig. 2. (a) Lemma 4. (b) Lemma 5.

If we can compute the longest-paths lengths $\lambda_R[v]$ to all vertices v in G^R, and thus the longest-path lengths $\lambda[v]$ to all separator vertices v in G^s (Lemma 4), we can compute the longest-path lengths to the remaining vertices in G^s using the following lemma:

Lemma 5. *Let u be a vertex in the cluster G_i of G^s and let $\lambda_{\overline{G_i}}(v,u)$ denote the length of the longest path from a boundary vertex $v \in \partial G_i$ to u in $\overline{G_i}$. The length of the longest path from s to u in G^s is*

$$\lambda[u] = \max\{1, \max_{v \in \partial G_i}\{\lambda[v] + \lambda_{\overline{G_i}}(v,u)\}\}$$

Proof. Let p be the longest path from s to u in G^s. Either p has length one (edge (s,u)) or it must contain a vertex on the boundary ∂G_i of G_i. Consider the last such vertex $v \in \partial G_i$ and let p_{sv} and p_{vu} denote the subpath of p from s to v and from v to u, respectively. Refer to Fig. 2(b). By Lemma 3, since p is the longest path to u and G^s is a DAG, p_{sv} must be the longest path from s to v in G^s and p_{vu} the longest path from v to u in $\overline{G_i}$. If follows that we can find the length of p (that is, $\lambda[u]$) by evaluating $\lambda[v] + \lambda_{\overline{G_i}}(v,u)$ for each vertex v on ∂G_i. □

We can easily compute G^R from G^s I/O-efficiently as follows: We load each of the $O(\frac{N}{B^2})$ graphs $\overline{G_i}$ induced by G_i and ∂G_i into main memory in turn, use an internal memory algorithm to compute the relevant $O(B^2)$ edges between the $O(B)$ boundary vertices ∂G_i, and write these edges back to disk. During this process we also compute the $O(N/B)$ edges incident to s and retain the $O(N/B)$ edges in G^s between the separator vertices. In total we use $O(\frac{N}{B^2} \cdot B + \text{scan}(N)) = O(\text{scan}(N))$ I/Os. In the subsequent subsections we will assume that G^R is represented similarly to the way G^s (and G) is represented, that is, as a list of edges such that all edges incident to each vertex are stored consecutively and, furthermore, such that edges incident to the vertices in each boundary set of G^s

are stored consecutively (even though vertices in G_i are removed from G^R we will still refer to the boundary sets of the vertices of G^R). We can easily produce this representation in a simple scan of the produced edges using another $O(\text{scan}(N))$ I/Os.

After having computed the longest-paths lengths $\lambda_R[v]$ to all vertices v in G^R, we can easily compute the longest-path lengths to the remaining vertices in G^s in $O(\text{scan}(N))$ I/Os using Lemma 5. We load each cluster G_i and its boundary vertices ∂G_i (now marked with longest path lengths) in turn, compute $\lambda_{\overline{G_i}}(v,u)$ for each pair of vertices $v \in \partial G_i$ and $u \in G_i$ using an internal memory algorithm and thus computing $\lambda[u]$, and finally write all the longest-path lengths back to disk. Overall we have proved the following:

Lemma 6. *Computing longest-path lengths for all vertices in the DAG G^s with $N+1$ vertices and $O(N)$ edges can be reduced in $O(scan(N))$ I/Os to computing longest-path lengths in the DAG G^R with $O(N/B)$ vertices and $O(N)$ edges.*

2.3 Reducing longest paths in G^R to topologically sorting of G^R

Computing longest-path lengths in G^R can easily be reduced to topologically sorting G^R (utilizing the ideas of a standard linear time algorithm for computing shortest paths in a DAG [10]). The basic observation is that if the last edge on the longest path p from s to a vertex u is (v,u), then the part of p from s to v is the longest path to v (Lemma 3). This means that if $(v_1,u),\ldots,(v_k,u)$ are the k in-edges of u and $w(v_i,u)$ the weight of edge (v_i,u), then $\lambda_R[u] = \max\{\lambda_R[v_1]+w(v_1,u), \lambda_R[v_2]+w(v_2,u),\ldots,\lambda_R[v_k]+w(v_k,u)\}$. Thus if we process and compute the longest paths of the vertices of G^R in topological order, we know that when processing vertex u we have already computed the longest paths to all in-neighbors v_i. We can therefore easily compute the longest path to u in a simple scan of its in-edges.

To implement the above algorithm I/O-efficiently, given G^R in topological order, we maintain a list L of the longest path lengths $\lambda[u]$ to all vertices u in G^R such that vertices in the same boundary set are stored consecutively. Recall that a boundary set is defined as a maximal subset of boundary vertices in G^s (and thus in G^R) that are adjacent to the same set of clusters G_i in G^s, and that edges incident to vertices in the same boundary set are stored consecutively in our representation of G^R. As we process a vertex u in the topological order, we scan its $O(B)$ in-edges from the representation of G^R and load the longest-paths lengths of all its $O(B)$ in-neighbors v_i from L in order to compute $\lambda_R[u] = \max\{\lambda_R[v_1]+w(v_1,u), \lambda_R[v_2]+w(v_2,u),\ldots,\lambda_R[v_k]+w(v_k,u)\}$; then we write $\lambda_R[u]$ back to L. Since G^R has $O(N/B)$ vertices and $O(N)$ edges, we use $O(\frac{N}{B} + \text{scan}(N)) = O(\text{scan}(N))$ I/Os to retrieve all vertices and scan their in-edges. To see that the $O(N)$ accesses to L can also be performed in $O(\text{scan}(N))$ I/Os, recall that each boundary set is of size $O(B)$ and is accessed once by each of its adjacent vertices in each of its adjacent clusters, that is, $O(B)$ times. Since the vertices in each boundary set are stored consecutively in L they can be loaded in $O(1)$ I/Os. Since there are $O(\frac{N}{B^2})$ boundary sets, the total number of I/Os

spent on accessing boundary sets from L is overall $O(B \cdot \frac{N}{B^2}) = O(\text{scan}(N))$. We have obtained the following:

Lemma 7. *Computing longest-path lengths in DAG G^R with $O(N/B)$ vertices and $O(N)$ edges can be reduced in $O(scan(N))$ I/Os to topologically sorting G^R.*

2.4 Topologically sorting G^R

Since G^R only has $O(N/B)$ vertices (and is obtained from a B^2-partition) we can topologically sort it I/O-efficiently using a slightly modified version of a standard topological sort algorithm [10]. We first compute the in-degree of each vertex in G^R. Next we number the vertices one at time while maintaining a list Q of indegree-zero vertices; initially the list contains only the source vertex s. We repeatedly remove and number an indegree-zero vertex v from Q; for each such vertex v we remove all edges of the form (v, u) from G^R (v's out-edges) by decrementing the indegree of u in L. When the indegree of a vertex u becomes zero we insert it in Q. It is easy to see that this algorithm correctly topologically sorts G^R.

The above algorithm can be performed I/O-efficiently as follows: The initial indegree of all vertices can be computed in $O(\text{scan}(N))$ I/Os in a simple scan of the representation of G^R. To number the vertices of G^R one-by-one efficiently, we again exploit the topology of the boundary sets of G^R: we maintain the indegrees in a list L such that the degrees of vertices in the same boundary set are stored consecutively in L. When processing an (indegree-zero) vertex v we load v and all its $O(B)$ neighbors from L, scan through the (out) edge-list of v while decrementing the relevant indegrees and writing them back to L. In total there are $O(N)$ accesses, one for each edge, but as in Section 2.3 we can argue (using that there are only $O(N/B^2)$ boundary sets) that they are performed in $O(N/B)$ I/Os. Thus we have obtained the following:

Lemma 8. *The DAG G^R with $O(N/B)$ vertices and $O(N)$ edges can be topologically sorted in $O(scan(N))$ I/Os.*

3 BFS and SSSP on planar DAGs

Given a topological order of the vertices of a general acyclic graph, SSSP (and thus BFS) can easily be solved in $O(\text{sort}(N))$ I/Os using an I/O-efficient priority queue. In this section we describe how this bound can be improved to $O(\text{scan}(N))$ for planar DAGs if a B^2-partition of the graph is given.

Our improved algorithm is essentially the same as our algorithm for computing longest paths described in Section 2, but modified to compute *shortest* paths rather than longest paths. Let s in G be the source vertex for the SSSP (BFS) problem. We reduce computing SSSP on G to computing SSSP on a reduced graph G^R, which we in turn reduce to computing a topological order on G^R. As previously, the key of the algorithm is that all these reductions can be performed

in $O(\text{scan}(N))$ I/Os and that the reduced graph G^R can be topologically sorted in $O(\text{scan}(N))$ I/Os (Lemma 8).

The reduced graph G^R is defined as follows. The vertices of G^R consist of the source vertex s and the separator vertices in G. The edges are defined as in Section 2.1, except that edge weights between vertices u, v on the boundary ∂G_i of G_i correspond to *shortest* path lengths in $\overline{G_i}$. The graph G_R is a DAG with $O(N/B)$ vertices and $O(N)$ edges and can be computed in $O(\text{scan}(N))$ I/Os given a B^2-partition of G. Using the same arguments as previously, it is easy to show that G^R preserves *shortest* paths in G, that is, that for any separator vertex v the length $\delta_G(v)$ of the shortest path between s and v in G is the same as the length $\delta_{G^R}(v)$ of the shortest path in G^R. Given that we can compute shortest paths $\delta_G(v)$ to all vertices in G^R, we can then compute the shortest paths to the remaining vertices u in G_i as $\delta_G(u) = \min_{v \in \partial G_i}\{\delta(v) + \delta_{\overline{G_i}}(v, u)\}$. This can be done using $O(\text{scan}(N))$ I/Os as in Section 2.2. Since G^R is a DAG, shortest paths in G^R can be computed in the same way as longest paths in G^R (Section 2.3) by processing vertices in topological order. Finally, a topological order of G^R can be computed in $O(\text{scan}(N))$ I/Os using Lemma 8. We have the following.

Theorem 2. *Given a B^2-partition of a planar DAG G in edge-list representation, BFS and SSSP can be solved in $O(\text{scan}(N))$ I/Os on G.*

4 Conclusion and open problems

In this paper we developed simple linear I/O algorithms for the single-source shortest path, breadth-first search and topological sorting problems on planar DAGs, provided that a B^2-partition is given. Our algorithms rely on a set of reductions using the partition and essentially exploits the acyclicity of the graph. This leads to $O(\text{sort}(N))$ I/O algorithms for the three problems that are much simpler than the previously known algorithms.

It remains an intriguing open problem to develop an $O(\text{sort}(N))$ directed DFS algorithm, on planar graphs as well as on general graphs. The results of this paper naturally open the question if acyclicity can be exploited to derive an efficient DFS algorithm for planar DAGs.

References

1. NASA Earth Observing System (EOS) project. `http://eos.nasa.gov/`.
2. A. Aggarwal and J. S. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
3. L. Arge, G. S. Brodal, and L. Toma. On external memory MST, SSSP and multi-way planar graph separation. In *Proc. Scandinavian Workshop on Algorithms Theory, LNCS 1851*, pages 433–447, 2000.
4. L. Arge, U. Meyer, L. Toma, and N. Zeh. On external-memory planar depth first search. *Journal of Graph Algorithms*, 7(2):105–129, 2003.

5. L. Arge, L. Toma, and J. S. Vitter. I/O-efficient algorithms for problems on grid-based terrains. *ACM Journal on Experimental Algorithmics*, 6(1), 2001.
6. L. Arge, L. Toma, and N. Zeh. I/O-efficient topological sorting of planar DAGs. In *Proc. ACM Symposium on Parallel Algorithms and Architectures*, pages 85–93, 2003.
7. L. Arge and N. Zeh. I/O-efficient strong connectivity and depth-first search for directed planar graphs. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 261-270, 2003.
8. A. L. Buchsbaum, M. Goldwasser, S. Venkatasubramanian, and J. R. Westbrook. On external memory graph traversal. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 859–860, 2000.
9. Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. External-memory graph algorithms. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 139–149, 1995.
10. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., second edition.
11. G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.
12. V. Kumar and E. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. In *Proc. IEEE Symp. on Parallel and Distributed Processing*, pages 169–177, 1996.
13. R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal of Applied Math.*, 36:177–189, 1979.
14. A. Maheshwari and N. Zeh. I/O-optimal algorithms for planar graphs using separators. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 372–381, 2002.
15. U. Meyer, P. Sanders, and J. F. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS*. Springer, 2003.
16. NASA Jet Propulsion Laboratory. NASA Shuttle Radar Topography Mission (SRTM). http://www.jpl.nasa.gov/srtm/.
17. L. Toma. *External Memory Graph Algorithms and Applications to Geographic Information Systems*. PhD thesis, Duke University, 2003.