I/O-Efficient Algorithms on Near-Planar Graphs

Herman Haverkort[1,⋆] and Laura Toma[2]

[1] Department of Computing Science, Eindhoven University of Technology,
PO Box 513, 5600 MB Eindhoven, The Netherlands
cs.herman@haverkort.net
[2] Department of Computer Science, Bowdoin College,
8650 College Station, Brunswick, ME 04011,
United States of America
ltoma@bowdoin.edu

Abstract. Obtaining I/O-efficient algorithms for basic graph problems on sparse directed graphs is a long-standing open problem. While the best known upper bounds for most basic problems on such graphs with V vertices still require $\Omega(V)$ I/Os, optimal $O(sort(V))$ I/O algorithms are known for special classes of sparse graphs, like planar graphs and grid graphs. It is hard to accept that a problem becomes difficult as soon as the graph contains a few deviations from planarity. In this paper we extend the class of graphs on which basic graph problems can be solved I/O-efficiently. We give a characterization of near-planarity which covers a wide range of near-planar graphs, and obtain the first I/O-efficient algorithms for directed graphs that are near-planar.

1 Introduction

When working with massive graphs, only a fraction of the data can be held in the main memory of a computer. Thus, the transfer of blocks of data between main memory and disk, rather than the internal memory computation, is often the bottleneck. Therefore, developing *external-memory* or *I/O-efficient algorithms*—algorithms that specifically optimize the number of block transfers between main memory and disk, can lead to considerable runtime improvements.

I/O-efficient algorithms for graph problems has been an active area of research. Even though significant progress has been made, there is still a significant gap between the lower and the upper bound for all basic problems. Consider a directed graph (digraph) with non-negative real edge weights. A shortest path from vertex u to vertex v in G is a minimum-length path from u to v in G, where the length of a path is the sum of the weights of the edges on the path. The *single-source-shortest-paths (SSSP)* problem is to find shortest paths from a source

⋆ Part of this work was done while Herman Haverkort was at Karlsruhe University, supported by the European Commission, FET open project DELIS (IST-001907), and subsequently at Aarhus University, supported by a grant from the Danish National Science Research Council.

J.R. Correa, A. Hevia, and M. Kiwi (Eds.): LATIN 2006, LNCS 3887, pp. 580–591, 2006.

vertex s to all vertices in G. For *planar* digraphs (graphs that can be embedded in the plane such that no two edges intersect), there exist SSSP-algorithms with upper bounds on the number of block transfers that match proven lower bounds up to a constant factor. However, for general graphs, the SSSP problem is still open, as are other basic problems such as connected components (CC) and depth- and breadth-first search (DFS, BFS).

Both from a theoretical and from a practical point of view, it is hard to accept that SSSP should become extremely difficult as soon as a graph contains a few deviations from planarity. In practice, networks (e.g. transportation networks) may not be planar. However, when edges are expensive and junctions are cheap, such networks still have a strong tendency to planarity: there will be only relatively few links (e.g. motorways) that cross other edges without connecting to them. Other examples are networks in which each vertex is connected to a few nearby vertices. In such networks, there may be quite a number of crossings but they are all very 'local'. In this paper we give a characterization of near-planarity covering a wide range of near-planar graphs, and develop the first I/O-efficient algorithms for such graphs.

I/O-Model and related work. We develop I/O-efficient algorithms using the standard two-level I/O-model [2]. The model defines two parameters: M is the number of vertices/edges that fit into internal memory, and B the number of vertices/edges that fit into a disk block, where $B \leq M/2$. An *Input/Output* (or: *I/O*) is the operation of transferring a block of data between main memory and disk. The *I/O-complexity* of an algorithm is the number of I/Os it performs. The basic bounds in the I/O-model are those for scanning and sorting. The *scanning bound*, $scan(N) = \frac{N}{B}$, is the number of I/Os necessary to read N contiguous items from disk. The *sorting bound*, $sort(N) = \Theta(\frac{N}{B} \log_{M/B} \frac{N}{B})$, represents the number of I/Os required to sort N contiguous items on disk [2] when $N > M$. For all realistic values of B and $M < N$, we have $scan(N) < sort(N) \ll N$.

I/O-efficient graph algorithms have been considered by a number of authors; for a recent review see [23]. On general digraphs $G = (V, E)$ the best known algorithm for SSSP, as well as for the BFS and DFS traversal problems, use $\Omega(V)$ I/Os in the worst case[1]; their complexity is $O(\min\{(V + \frac{E}{B}) \cdot \log V + sort(E), V + \frac{V}{M}\frac{E}{B}\})$ [12, 13, 19]. On sparse graphs, which have $E = O(V)$, the best known bounds are thus $O(V)$ I/Os or worse, which is no better than just running the internal-memory algorithms in external memory. This is far from the currently best lower bound of $\Omega(\min\{V, sort(V)\} + E/B)$ I/Os, which on sparse graphs is practically $\Omega(sort(V))$.

The search for BFS, DFS and SSSP algorithms using $O(sort(E))$ I/Os on general (sparse) graphs has led to a number of improved results for special graph classes [5, 6, 7, 8, 10]. All these algorithms are based on the existence of small separators. For planar graphs, they exploit graph partitions, as introduced by Frederickson [16]. For any planar graph $\mathcal{K} = (V, E)$, given a parameter $R \leq V$, we can find a subset $V_S \subset V$ of $O(V/\sqrt{R})$ vertices, such that the removal of V_S

[1] We denote the size of a set by its name; the meaning will be clear from the context.

partitions $\mathcal{K}$ into subgraphs $\mathcal{K}_i$ such that: (1) there are $O(V/R)$ subgraphs; (2) each subgraph has size $O(R)$, and (3) (the vertices in) each $\mathcal{K}_i$ is (are) adjacent to $O(\sqrt{R})$ vertices of V_S. We call such a partition an *R-partition*. Assuming that $R \leq M/(c \log^2 B)$, for a sufficiently big constant c, an R-partition can be computed I/O-efficiently with $O(sort(V))$ I/Os [22]. On planar digraphs, using R-partitions, SSSP and BFS can be solved in $O(sort(V))$ I/Os [8], and DFS in $O(sort(V) \log \frac{V}{M})$ I/Os [10];

Our results. In this paper we extend the class of graphs that admit I/O-efficient algorithms. We introduce a class of near-planar graphs and show how to find small separators for planar subgraphs of such graphs that gracefully depend on the non-planarities. Using these separators, we develop the first I/O-efficient SSSP, BFS, DFS and topological sort algorithms for such near-planar graphs.

Our main result is the following. Let $G = (V, E \cup E_C)$ be a digraph that consists of a planar graph $\mathcal{K} = (V, E)$ and a given set of additional edges E_C; let $G_C = G - \mathcal{K} = (V_C, E_C)$ denote the non-planar part of G, where V_C is the set of vertices incident to edges in E_C. We show how to refine an R-partition of $\mathcal{K}$ to restrict the number of vertices of V_C per subgraph, while adding no more than $O(\sqrt{VV_C}/R^{1/4})$ vertices to the separator and increasing the number of subgraphs by no more than $O(V_C/\sqrt{R})$. Using refined R-partitions we show how to compute SSSP on G in $O(E_C + sort(V + E_C))$.

We generalize our result to graphs $G = (V, E \cup E_C)$ such that $\mathcal{K} = (V, E)$ can be drawn in the plane with T crossings. If we know for each edge (u, v) of $\mathcal{K}$ which edges it crosses, and in which order these crossings occur when traversing the edge from u to v, we can compute SSSP on such a graph G in $O(E_C + sort(V + T + E_C))$ I/Os.

When a graph is *near-planar* in the sense that $T = O(V)$ and $E_C = O(V/B)$, these bounds reduce to $O(sort(V))$, whereas the best known SSSP-algorithm for general graphs requires $O((V + \frac{E}{B}) \cdot \log \frac{V}{B} + sort(E)) \supset O(V)$ I/Os. If information about a suitable drawing of a graph is given, our results allow the computation of SSSP in $O(sort(E))$ I/Os on graphs with crossing number $O(E)$, on graphs that are k-embeddable in the plane for constant k, on graphs with skewness $O(E/B)$ and on graphs with splitting number $O(E/B)$. We obtain similar results for BFS, DFS, topological order and CC.

Outline. The paper is organized as follows. Sec. 2 presents refined R-partitions and Sec. 3 describes how to use these partitions to compute SSSP efficiently. Sec. 4 extends our approach to other basic graph problems. In Sec. 5 we explain how our technique could be used for problems on several types of graphs that are near-planar according to measures of planarity proposed in literature. We conclude in Sec. 6 and give directions for further research.

2 Partitioning a Near-Planar Graph

In this section we discuss how to compute small separators and extend Frederickson's R-partitions to graphs that are not planar. Consider a graph

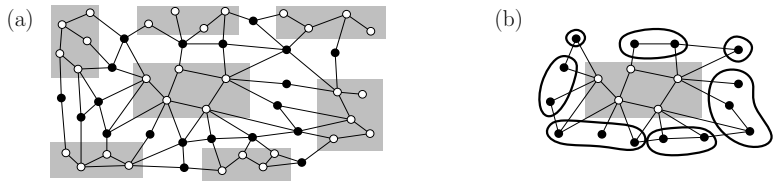

Fig. 1. (a) Partition of a planar graph into clusters (boxed) and separator vertices (black). (b) One cluster in the partition and its adjacent boundary sets.

$G = (V, E \cup E_C)$ that consists of planar subgraph $\mathcal{K} = (V, E)$ and a set of edges E_C. For this section we assume $\mathcal{K}$ to be known. Let $G_C = (V_C, E_C) = G - \mathcal{K}$ denote the non-planar part of G. We call the edges of G_C *cross-link edges*, and the vertices of G_C *cross-link vertices*. We assume that the vertices and edges in the *cross-link graph* G_C are labeled as such.

We start by computing an R-partition for $\mathcal{K} = (V, E)$, that is, a set $V_S \subset V$ of $O(V/\sqrt{R})$ vertices, such that the removal of V_S partitions $\mathcal{K}$ into subgraphs $\mathcal{K}_i$ such that there are $O(V/R)$ subgraphs, each subgraph has size $O(R)$, and is adjacent to $O(\sqrt{R})$ vertices of V_S. We use the following notation: the vertices in V_S are *separator vertices* and each of the subgraphs a *cluster*; the set of vertices in $\mathcal{K} - \mathcal{K}_i$ adjacent to $\mathcal{K}_i$ are the *boundary vertices* $\partial\mathcal{K}_i$ (or simply the *boundary*) of $\mathcal{K}_i$. We use $\overline{\mathcal{K}_i}$ to denote the graph consisting of $\mathcal{K}_i$, $\partial\mathcal{K}_i$ and the subset of edges of E connecting vertices in $\mathcal{K}_i \cup \partial\mathcal{K}_i$. The set of separator vertices can be partitioned into maximal subsets so that the vertices in each subset are adjacent to precisely the same set of clusters. These sets are the *boundary sets* of the partition. If the graph has bounded degree, which can be ensured for planar graphs using a simple transformation, there exists an R-partition with only $O(V/R)$ boundary sets [16] (Refer to Fig. 1).

The separator V_S is a separator for $\mathcal{K}$ but not necessarily for G, because any cluster in $\mathcal{K}$ may contain up to R cross-link vertices that are connected by cross-link edges to cross-link vertices in other clusters, by-passing the separator. Let G_i denote the clusters induced by $\mathcal{K}_i$ in G. A straightforward way to get a separator for G would be to add all cross-link vertices V_C to V_S; however, the SSSP algorithm of Sec. 3, run on the basis of such a separator, would use $O(E_C + V)$ I/Os.

We show how to refine the partition of $\mathcal{K}$ to incorporate the cross-link edges while ensuring that the total number of separator vertices and clusters is not too large and each cluster contains $O(\sqrt{R})$ cross-link vertices. Our approach is based on the following generalization of Lemma 2 from [16].

Lemma 1. *Given a subgraph $G = (V, E)$ of a planar graph with $|\partial G| = O(\sqrt{V})$, and a weight function $w : V \to \mathbb{R}$ such that $\sum_{v \in V} w(v) = W$, we can find a subset $S \subset V$ of size $O(\sqrt{VW})$ which separates $G - S$ into a set of $O(W)$ subgraphs (clusters) G' with the following properties:*

- *each cluster $G' = (V', E')$ has a total weight $\sum_{v \in V'} w(v)$ of at most 1.*
- *for each cluster $G' = (V', E')$, we have that $\partial G'$ has $O(\sqrt{V})$ vertices.*

Proof. The proof follows the proof of Lemma 1 and 2 from Frederickson [16], which is based on recursive application of the separator theorem by Lipton and Tarjan [21] in two phases: first with uniform weights on the vertices, and then with weights on the separator vertices only. However, we use a non-uniform weight function in the first phase. Note that we are not interested in low-weight separators: it is the weights of the clusters that count. The first phase of the recursive procedure is as follows. When G has weight $w(G)$ at most 1, we are done. Otherwise, applying Lipton and Tarjan's separator theorem, we find a subset S of at most $2\sqrt{2}\sqrt{V}$ vertices of V such that S separates $G - S$ into two clusters A and B that each have weight at most $\frac{2}{3}w(G)$. We partition the clusters A and B recursively.[2] This procedure results in a number of clusters. By construction each cluster $G' = (V', E')$ has weight at most 1, and the number of clusters is obviously $O(W)$. However, the boundary $\partial G'$ of a cluster G' may still have more than $O(\sqrt{V})$ vertices—this is solved by the second phase. But first we show that so far, the total number of vertices in the subsets S that were selected is $O(\sqrt{VW})$. Let $s(v, w)$ be the maximum number of separator vertices that may be selected while recursively partitioning a planar graph induced by a set of v vertices with weight w. Note that any of its subgraphs A and B may have total weight at most $\frac{2}{3}w$, and at least one of them has at most $v/2$ vertices. Therefore $s(v, w)$ is bounded by the following recursive expression: $s(v, w) \leq \max_{0<\alpha\leq 1/2, 1/3\leq\beta\leq 2/3} c\sqrt{v} + s(\alpha v, \beta w) + s((1-\alpha)v, (1-\beta)w)$ where $s(v, w) = 0$ if $w \leq 1$, and $c = 2\sqrt{2}$. This recursion solves to $s(V, W) = O(\sqrt{VW})$ (details in the full version of this paper). In the second phase of the procedure we recursively subdivide each cluster further until the size of its boundary is reduced to $O(\sqrt{V})$. It can be shown that this increases the number of separator vertices and the number of clusters by at most a constant factor; thus the lemma follows. □

Our algorithm first computes an R-partition of $\mathcal{K}$ in $O(sort(V))$ I/Os with the algorithm by Maheshwari and Zeh [22]; then we refine the partition by applying Lemma 1 to each cluster $\overline{G_i}$ that has more than $c\sqrt{R}$ cross-link vertices, for some fixed constant c. For each such cluster we assign weight $1/(c\sqrt{R})$ to every cross-link vertex in $\overline{G_i}$ and weight 0 to every other vertex. Thus each cluster that results from refining $\overline{G_i}$ has $O(\sqrt{R})$ cross-link vertices, $O(R)$ vertices in total, and $O(\sqrt{R})$ vertices on its boundary.

We use Lemma 1 to bound the number of separator vertices and number of clusters G' resulting from the refinement. Cluster $\overline{G_i}$ has total weight $W_i = \sum_{v\in\overline{G_i}} w(v) = |\overline{G_i} \cap V_C|/(c \cdot \sqrt{R})$. For each cluster G_i, the number of separator vertices obtained by refining it is $O((|\overline{G_i}| \cdot W_i)^{1/2}) = O(R^{1/4}(|\overline{G_i} \cap V_C|)^{1/2})$. Summed over all clusters G_i this adds $O(R^{1/4}\sum_{G_i}(|\overline{G_i} \cap V_C|)^{1/2})$ separator vertices in total. Since $2\sqrt{(a+b)/2} \geq \sqrt{a} + \sqrt{b}$, the worst case occurs if the cross-link vertices V_C are evenly distributed over the $O(V/R)$ subgraphs G_i, and we get: $R^{1/4}\sum_{G_i}(|\overline{G_i}\cap V_C|)^{1/2} \leq R^{1/4}O(V/R)O(\sqrt{V_C R/V}) = O(V/R^{3/4} + \sqrt{VV_C}/R^{1/4})$. Adding this to the $O(V/\sqrt{R})$ vertices that were already in V_S before we started refining the partition, we get a total of $O(V/\sqrt{R} + \sqrt{VV_C}/R^{1/4})$.

[2] Alternatively, one could apply the results of Aleksandrov et al. [3] for the first phase.

Similarly, the number of clusters obtained by refining each $\overline{G_i}$ is $O(W) = O(|\overline{G_i} \cap V_C|/(c \cdot \sqrt{R}))$ (by Lemma 1), and we can show that the total number of clusters is $O(V/R + V_C/\sqrt{R})$. Overall we have the following (due to space constraints the complete proof is omitted):

Theorem 1. *Let R be a parameter such that $R \leq M/(c \log^2 B)$, for a sufficiently big constant c. We can, with sort(E) I/Os, find a subset $V_S \subset V$ whose removal separates $\mathcal{K}$ into a set of subgraphs G_i with the following properties:*

- *the total number of vertices in V_S is $O(V/\sqrt{R} + \sqrt{VV_C}/R^{1/4})$*
- *there are $O(V/R + V_C/\sqrt{R})$ subgraphs G_i in $\mathcal{K} - V_S$*
- *each subgraph contains $O(R)$ vertices, is adjacent to $O(\sqrt{R})$ separator vertices and contains $O(\sqrt{R})$ cross-link vertices.*

3 Computing SSSP Using the Refined *R*-Partition

We now show how to use the refined partition of a non-planar graph G obtained in Sec. 2 above to compute SSSP I/O-efficiently.

The standard approach used by I/O-efficient planar graph algorithms is as follows. Given an R-partition of a planar graph $\mathcal{K}$, we compute a substitute graph $\mathcal{K}^R$ defined on the separator vertices. The graph $\mathcal{K}^R$ is a *reduced* version of $\mathcal{K}$ (it has fewer vertices), and we construct it such that the lengths of the shortest paths in $\mathcal{K}^R$ are the same as in $\mathcal{K}$. The SSSP algorithm consists of three steps: (1) Compute $\mathcal{K}^R$; (2) Compute SSSP in $\mathcal{K}^R$ (by construction, we know these are the lengths of the shortest paths in $\mathcal{K}$); (3) Compute the shortest paths to vertices inside the clusters $\mathcal{K}_i$ of the R-partition.

To extend this approach to a non-planar graph G, we have to incorporate the cross-link (non-planar) edges E_C of G. We do this on the basis of a refined R-partition of G that divides G into subgraphs G_i, as explained in Sec. 2. Note that a shortest path between two arbitrary vertices in G enters and exits a subgraph $\overline{G_i}$ either through a boundary vertex or through a cross-link vertex. Therefore the substitute graph G^R will be defined on both the separator *and* the cross-link vertices and it contains an edge between each cross-link vertex and the boundary vertices of its cluster. Since this introduces $O(V_C\sqrt{R})$ edges in G^R, care must be taken so that the number of I/Os spent on them does not become $\Omega(V_C\sqrt{R})$.

Below we show how to exploit Theorem 1 to implement the substitute graph of a refined R-partition of a non-planar graph such that shortest paths can be computed efficiently. We will give details and prove this section's main result:

Theorem 2. *SSSP on a digraph $G = \mathcal{K} \cup G_C$ uses $O(E_C + sort(V + E_C))$ I/Os.*

3.1 The Substitute Graph

We obtain G^R as follows: First, it includes the edges between the separator vertices in the partition (that is, in G), and the edges between the cross-link vertices, i.e. the cross-link graph G_C. Second, it includes the union of all complete graphs G_i^R obtained by replacing each subgraph $\overline{G_i}$ as follows: the vertices of

G_i^R are the boundary vertices ∂G_i of G_i and the cross-link vertice $V_C \cap G_i$ of G_i, and there is an edge from u to v in G_i^R if there is a path from u to v in $\overline{G_i}$. The edge (u, v) has weight equal to the length of the shortest path from u to v in $\overline{G_i}$. Note that G_i^R contains edges between boundary vertices, between cross-link vertices and boundary vertices, and between cross-link vertices. Third, if the SSSP source vertex s is not a separator or a cross-link vertex, we add it to G^R and add edges from s to all the boundary vertices and all cross-link vertices of the subgraph G_i containing s; as above, the weight of an edge (s, v) is the length of the shortest path from s to v in $\overline{G_i}$.

Let $\delta_G(u, v)$ denote the shortest path from u to v in G. For any pair of vertices $u, v \in V_S \cup V_C \cup \{s\}$ we can show that $\delta_{G^R}(u, v) = \delta_G(u, v)$, that is, G^R maintains shortest paths between its vertices. The number of vertices in the substitute graph is $V_S + V_C + 1$, which, by Theorem 1, is $O(V/\sqrt{R} + \sqrt{VV_C}/R^{1/4} + V_C)$. By Theorem 1, there are $O(V/R + V_C/\sqrt{R})$ subgraphs in total, each of which has $O(\sqrt{R})$ boundary vertices, $O(\sqrt{R})$ cross-link vertices, and possibly a source vertex; thus each complete graph G_i^R has $O(R)$ edges in total. In total $\cup G_i^R$ has $O(V/R + V_C/\sqrt{R}) \cdot O(R) = O(V + V_C\sqrt{R})$ edges. Add the $O(V/\sqrt{R} + \sqrt{VV_C}/R^{1/4} + E_C)$ cross-link edges and edges between separator vertices in the partition, and we get:

Lemma 2. *The substitute graph G^R has $O(V/\sqrt{R} + \sqrt{VV_C}/R^{1/4} + V_C)$ vertices and $O(V + V_C\sqrt{R} + E_C)$ edges.*

We can also show that G^R can be computed in $O(scan(E) + sort(|G^R|))$. We defer the details to the full version of this paper.

3.2 Computing SSSP on G^R

To compute SSSP on G^R we use Dijkstra's algorithm, which we make I/O-efficient by modifying it to take advantage of the structure of G^R. In addition to a priority queue, we maintain a list L that stores the tentative distances from s to all the vertices in G^R, that is, in $V_S \cup V_C \cup \{s\}$. When extracting a vertex from the priority queue, we retrieve the tentative distances of its out-neighbors from L. For each out-neighbor w of v we check whether its tentative distance as stored in L is greater than $d(v)$ plus the weight of the edge (v, w); if it is, we update the distance of w in L, delete the old entry of w from the priority queue and insert a new entry for w with the updated distance in the queue.

In total, we perform $O(V(G^R)) = O(V_S + V_C)$ `ExtractMin`s, and $O(E(G^R)) = O(V + V_C\sqrt{R} + E_C)$ `Delete`s and `Insert`s on the priority queue. These operations can be performed efficiently in $O(sort(V + V_C\sqrt{R} + E_C))$ I/Os using an I/O-efficient priority queue, e.g. [4]. We also perform $O(E(G^R)) = O(V + V_C\sqrt{R} + E_C)$ accesses to the list L; this is because every vertex in L is accessed once by each incoming edge in G^R. Of course, we cannot afford one I/O per edge. In order to perform the accesses to L efficiently, we store L in the following order: all vertices in V_S are at the front of L, grouped by boundary set, followed by the vertices in $V_C - V_S$, grouped by the index of the subgraph G_i that contains them. Note

that with this order the vertices in the same boundary set, as well as cross-link vertices in the same cluster, are consecutive in L.

Lemma 3. *The accesses to the list L can be performed in $O(V_S+E_C+(V/\sqrt{R}+V_C)\cdot\lceil\sqrt{R}/B\rceil)$ I/Os.*

Proof. The accesses to the list L are of three types: (1) $O(E_C)$ accesses through the cross-link edges of G^R; (2) $O(V_S)$ accesses through edges between separator vertices; and (3) $O(V + V_C\sqrt{R})$ accesses through the edges in the substitute graphs G_i^R. The first two types of accesses clearly take $O(V_S+E_C)$ I/Os. We now analyze the third type of accesses to L by counting the number of accesses per boundary set (while ignoring the cross-link edges, which are counted separately in (1)). Recall that a boundary set is a maximal set of separator vertices which are adjacent to precisely the same subgraphs G_i. Every vertex $v \in V_S \cup V_C \cup \{s\}$ in G^R that is processed needs to access the tentative distances of its out-neighbors in L: that is, every separator vertex $v \in V_S$ needs to access all the boundary vertices and cross-link vertices of all subgraphs G_i adjacent to v; every vertex $v \in \{s\} \cup V_C \setminus V_S$ needs to access all the boundary vertices and all cross-link vertices in the subgraph G_i containing v. Every time a vertex in a boundary set needs to be accessed, the other vertices in the boundary set need to be accessed as well, since the vertices of a boundary set are adjacent to the same subgraphs. For simplicity, we can think of all the cross-link vertices in a subgraph G_i as an additional "boundary" set of that subgraph. Overall, each boundary set of G^R is accessed once by each of the vertices on the boundaries of the subgraphs adjacent to the boundary set, and by each of the cross-link vertices in these subgraphs. By Theorem 1, each subgraph G_i has $O(\sqrt{R})$ boundary and $O(\sqrt{R})$ cross-link vertices. Thus each boundary set is accessed $O(\sqrt{R})$ times for each adjacent subgraph.

By the planar graph argument [16] the number of boundary sets as well as the number of adjacencies between boundary sets and subgraphs G_i is asymptotically the same as the number of subgraphs G_i. Using Theorem 1 we get that the total number of accesses to boundary sets is $O(\sqrt{R}) \cdot O(V/R + V_C/\sqrt{R}) = O(V/\sqrt{R} + V_C)$. Since boundary sets are stored consecutively in L (including the "boundary" set consisting of the $O(\sqrt{R})$ cross-link vertices of a subgraph), each boundary set can be accessed in $\lceil\sqrt{R}/B\rceil$ I/Os.

Thus the accesses to boundary sets use in total $O(V/\sqrt{R}+V_C)\cdot\lceil\sqrt{R}/B\rceil$ I/Os. Adding the $O(V_S)$ accesses between separator vertices and the $O(E_C)$ I/Os to L caused by the cross-link edges (type (1) and (2)), we get a total of $O(V_S + E_C + (V/\sqrt{R}+V_C)\cdot\lceil\sqrt{R}/B\rceil)$ I/Os. □

Putting together the operations on the priority queue and the accesses to the list L (Lemma 3) we get that computing SSSP on G^R uses $O(V_S+E_C+(V/\sqrt{R}+V_C)\cdot\lceil\sqrt{R}/B\rceil + sort(V + V_C\sqrt{R} + E_C))$ I/Os.

The third step in the SSSP algorithm on G computes shortest paths to all vertices in $V - (V_S \cup V_C)$. In the full paper we show that this step is dominated by the previous two steps. From the above we get that the total number of I/Os to compute SSSP on G is $O(sort(V + V_C\sqrt{R} + E_C) + V_S + E_C + (V/\sqrt{R}$

$+V_C)\lceil\sqrt{R}/B\rceil)$, which is $O(V/\sqrt{R}+\sqrt{VV_C}/R^{1/4}+E_C+sort(V+V_C\sqrt{R}+E_C))$. Assume for simplicity that $M > B^2$. If $V_C < V/B$, we choose $R = B^2$ and the bound becomes $O(E_C + sort(V + E_C))$. If $V_C > V/B$, we choose $R = (V/V_C)^2 = O(M)$ and again get $O(E_C + sort(V + E_C))$. This concludes the proof of Theorem 2.

4 Other Graph Problems Using Refined Partitions

The ideas from the SSSP algorithm above can be extended to other algorithms on near-planar graphs. We mention results for connected components (CC), topological order and depth-first search (DFS) and leave details for the full version.

Theorem 3. *Let $G = \mathcal{K} \cup G_C$. A topological order (assuming G is a DAG) and the connected components of G (assuming G is undirected) can be computed with $O(E_C + sort(V + E_C))$ I/Os. A DFS ordering can be computed with $O(V/\sqrt{B} + E_C)$ I/Os.*

5 Planarizing Graphs

The question how close a given graph is to being planar, is much-studied and has obvious applications in, for example, graph drawing and in the manufacturing of VLSI circuits. Several generalizations of planarity and measures of planarity have been defined, including crossing number, k-embeddability in the plane, skewness, splitting number and thickness—for a survey, see Liebers [20]. The class of near-planar graphs studied in this paper includes graphs which have low crossing number, are k-embeddable for small k, have low skewness, or have low splitting number—provided information about a suitable drawing of the graph is given. We will now briefly review these measures of planarity and discuss how a near-planar graph can be preprocessed so that it can be operated on by the algorithms described in the previous sections of this paper.

Graphs with low crossing number. The crossing number of a graph $G = (V, E)$ is the minimum number of edge crossings needed in any drawing of a given graph in a plane. When a drawing with T crossings is given, it can be preprocessed so that our SSSP algorithm described in the previous sections uses $O(sort(E+T))$ I/O's. The idea is to represent each crossing i by a vertex $v(i)$, which is marked as a *crossing*. Each crossed edge (u, u'), with crossings $i_1, ..., i_n$ in order going from u towards u', is replaced by edges $(u, v(i_1)), (v(i_1), v(i_2)), ..., (v(i_{n-1}), v(i_n))$, $(v(i_n), u')$. The transformation can easily be carried out in $O(sort(E+T))$ I/O's.

The resulting graph is a planar graph with $O(V)$ original vertices and $O(T)$ crossing vertices, where the crossing vertices have the special property that shortest paths are not allowed to turn on such vertices. The partitioning scheme and SSSP algorithm described in the previous sections can easily be adapted to work on graphs in which some of the vertices represent such crossings. We start

by applying the partitioning scheme as usual, ignoring the fact that some vertices represent crossings. After computing the refined separator V_S, we remove the crossing vertices and restore the original connectivity of the graph. When done carefully, this may make clusters and boundary sets non-planar, but it will not affect which boundary sets are adjacent to which clusters. Thus the SSSP algorithm will still work correctly within the claimed I/O-bounds, requiring $O(sort(V'))$ I/O's on such a graph, where $V' = O(V + T)$.

A graph is k-embeddable in the plane if it can be drawn in the plane so that each edge crosses at most k other edges [24]. Since a k-embeddable graph necessarily has small crossing number, the above approach can be taken.

Graphs with low skewness. The skewness of a graph $G = (V, E)$ is the minimum size of any set of edges E_C such that $G \setminus E_C$ is planar. When the skewness of a graph is $O(E/B)$ and E_C is given, our SSSP algorithm needs only $O(sort(E))$ I/Os, even if the edges and vertices in E_C form a clique with crossing number $\Theta(E^2/B^2)$.

When E_C is not given, it may be difficult to find it. Finding a minimum-size set E_C corresponds to finding a maximum-size planar subgraph of G. These are NP-complete problems [17]. When a drawing of the graph is given, we can define a *crossing graph* $G' = (V', E')$ in which V' has a vertex $v(e)$ for every edge e in G, and E' has an edge $(v(e), v(f))$ for every pair of crossing edges e and f in G. Finding a factor-two approximation of a minimum-size set E_C such that the drawing of $G \setminus E_C$ is intersection-free can be expressed as a maximal-matching problem in G', which can be solved with the randomized algorithm by Abello et al. [1]. This takes $O(sort(E')) = O(sort(T))$ I/Os (expected), where T is the number of crossings in the input graph.

Although theoretically, this transformation is not any cheaper than the one described in the previous section, it may still be advantageous because the resulting planar graph with added cross-links may be a lot smaller than a graph in which crossings are replaced by auxiliary vertices.

Graphs with small splitting number. Splitting a vertex is the process of replacing a vertex u by two vertices u_1, u_2, whereby some of the edges incident to u will be reconnected to u_1, while the remaining edges incident to u are reconnected to u_2. The splitting number of a graph is the minimum number of splittings that is needed to make the graph planar.

When the splitting number of a graph is $O(E/B)$ and the necessary splittings are given, we can solve the SSSP problem on such a graph in $O(sort(E))$ I/Os, using an approach similar to that for graphs with small skewness. Instead of running the shortest-paths algorithm on the original graph, we run it on the planar graph resulting from the splittings, augmented with a zero-weight bidirectional cross-link (u_1, u_2) for every vertex u split into u_1 and u_2.

Combining crossings and cross-links. Above we mentioned that graphs that have low crossing number can be handled efficiently by replacing crossings by special vertices, while graphs with small skewness or small splitting number can be handled efficiently by identifying a small number of cross-link edges. The two

approaches can be combined: we can find shortest paths in $O(sort(E))$ I/Os on a graph that consists of $O(E/B)$ cross-links and a graph with crossing number $O(E)$, provided the cross-links and the intersections in the remaining graph are given. How to find a constant-factor approximation of a minimum-size set of cross-links such that the rest of the graph has crossing number $O(E)$, still remains as an open problem.

6 Discussion

In this paper we extended the class of graphs for which efficient SSSP computations are possible from planar graphs to several classes of near-planar graphs. Our approach yields efficient algorithms for graphs with low crossing number, low splitting number or low skewness, provided suitable drawings are given. In theory, creating suitable drawings is difficult, since identifying a maximum planar subgraph or computing the crossing number, splitting number or the skewness of a graph are NP-complete problems [15, 18, 25]. However, in many practical applications of graph algorithms, graphs are given with a drawing or suitable drawings can be produced by heuristic methods.

Even if a good drawing is given, the method to identify cross-links in a graph of low skewness as described in Sec. 5 needs to know all crossings in the drawing. The crossings would need to be given or would need to be computed: in the case of a rectilinear drawing[3] we could do so with the external-memory line segment intersection algorithm by Arge et al. [9] or the randomized algorithm by Crauser et al. [14]. One could hope to find an algorithm that can find an effective set of cross-links without computing all crossings in the drawing first. It would also be interesting to find a constant-factor approximation of a minimum-size set of cross-links such that the rest of the graph has crossing number $O(E)$, so that we may have only very few cross-links and handle the remaining crossings with auxiliary vertices as described in Sec. 5.

Furthermore, it would be interesting to look into more measures of planarity that may be exploited, for example thickness: the minimum number of planar subgraphs whose union is the original graph.

References

1. J. Abello, A. L. Buchsbaum, and J. R. Westbrook. A functional approach to external graph algorithms. *Algorithmica*, 32(3):437–458, 2002.
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. Aleksandrov, H. Djidjev, H. Guo, and A. Maheshwari. Partitioning planar graphs with costs and weights. In *Proc. Workshop on Algorithm Engineering and Experimentation*, volume 2409 of *LNCS*, pages 98–110, 2002.

[3] A rectilinear drawing in itself already poses limitations. The minimum number of crossing required in a *rectilinear* drawing of the graph may be arbitrarily much greater than the crossing number for drawings with curves [11].

4. L. Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003.
5. L. Arge, G. S. Brodal, and L. Toma. On external memory MST, SSSP and multiway planar graph separation. *Journal of Algorithms*, 53(2):186–206, 2004.
6. 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.
7. L. Arge and L. Toma. Simplified external-memory algorithms for planar DAGs. In *Proc. Scandinavian Workshop on Algorithm Theory*, pages 493–503, 2004.
8. L. Arge, L. Toma, and N. Zeh. I/O-efficient topological sorting of planar DAGs. In *Proc. ACM Symposium on Parallel Algorithms and Architectures*, 2003.
9. L. Arge, D. E. Vengroff, and J. S. Vitter. External-memory algorithms for processing line segments in geographic information systems. In *Proc. Eur. Symp. Algorithms*, volume 979 of *LNCS*, pages 295–310, 1995.
10. L. Arge and N. Zeh. I/O-efficient strong connectivity and depth-first search for directed planar graphs. In *Proc. IEEE Symp. on Found. of Computer Sc.*, 2003.
11. D. Bienstock and N. Dean. Bounds for rectilinear crossing numbers. *Journal of Graph Theory*, 17:333–348, 1993.
12. A. L. Buchsbaum, M. Goldwasser, S. Venkatasubramanian, and J. R. Westbrook. On external memory graph traversal. In *Proc. Symposium on Discrete Algorithms*, pages 859–860, 2000.
13. Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. External-memory graph algorithms. In *Proc. Symposium on Discrete Algorithms*, pages 139–149, 1995.
14. A. Crauser, P. Ferragina, K. Mehlhorn, U. Meyer, and E. Ramos. Randomized external-memory algorithms for some geometric problems. In *Proc. ACM Symposium on Computational Geometry*, pages 259–268, 1998.
15. L. Faria, C. M. H. de Figueiredo, and C. F. X. de Mendon¸ca Neto. Splitting number is np-complete. *Discrete Applied Mathematics*, 108:65–83, 2001.
16. G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.
17. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H Freeman & Co, 1979.
18. M. R. Garey and D. S. Johnson. Crossing number is np-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4:312–316, 1983.
19. V. Kumar and E. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. In *Proc. IEEE Symposium on Parallel and Distributed Processing*, pages 169–177, 1996.
20. A. Liebers. Planarizing graphs—a survey and annotated bibliography. *Journal of Graph Algorithms and Applications*, 5(1):1–74, 2001.
21. R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal of Applied Math.*, 36:177–189, 1979.
22. A. Maheshwari and N. Zeh. I/O-optimal algorithms for planar graphs using separators. In *Proc. Symposium on Discrete Algorithms*, pages 372–381, 2002.
23. U. Meyer, P. Sanders, and J. F. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS*. Springer, 2003.
24. J. Pach and G. Tóth. Graphs drawn with few crossings per edge. *Combinatorica*, 17:427–439, 1997.
25. T. Watanabe, T. Ae, and A. Nakamura. On the np-hardness of edge-deletion and -contraction problems. *Discrete Applied Mathematics*, 6:63–78, 1983.