External Data Structures for Shortest Path Queries on Planar Digraphs

Lars Arge[1,⋆] and Laura Toma[2]

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

Abstract. In this paper we present space-query trade-offs for external memory data structures that answer shortest path queries on planar directed graphs. For any $S = \Omega(N^{1+\epsilon})$ and $S = O(N^2/B)$, our main result is a family of structures that use S space and answer queries in $O(\frac{N^2}{SB})$ I/Os, thus obtaining optimal space-query product $O(N^2/B)$. An S space structure can be constructed in $O(\sqrt{S} \cdot \text{sort}(N))$ I/Os, where $\text{sort}(N)$ is the number of I/Os needed to sort N elements, B is the disk block size, and N is the size of the graph.

1 Introduction

Let $G = (V, E)$ be a directed graph (digraph) with real edge weights. If G has no negative-weight cycles, the shortest path $\delta(s, t)$ from vertex s to vertex t is the minimum length path from s to t in G, where the length of a path is defined as the sum of the weights of its edges. The length of the shortest path $\delta(s, t)$ is called the *distance* from s to t in G. Shortest path computation is a fundamental and well-studied problem that appears in a diverse set of applications. In recent years, an increasing number of these applications involve massive graphs. Massive planar graph problems and in particular shortest paths computation arise frequently in Geographic Information System (GIS), where datasets such as the ones acquired by missions like NASA Earth Observing System (EOS) or Space Radar Topography Mission (SRTM) are on the order of terabytes. Environmental researchers often need to compute shortest paths for instance when planning and assessing the impact of new development, or modeling the communication between areas of conservation for endangered species. When working with such massive graphs that do not fit in the main memory of even state-of-the-art machines, transfer of data between main memory and external memory (such as disk), rather than internal computation time, is often the performance bottleneck. In such cases it is important to consider algorithms that minimize Input-Output (or simply I/O) communication.

⋆ 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.

X. Deng and D. Du (Eds.): ISAAC 2005, LNCS 3827, pp. 328–338, 2005.

The most commonly studied shortest path problems are the *single-source-shortest-path* (SSSP) problem and the *all-pair-shortest-path* (APSP) problem, where the goal is to find the shortest paths from a source vertex s to all other vertices in G, and between all pairs of vertices in G, respectively. Several authors have considered I/O-efficient algorithms for these problems. In this paper we study another variant of the problem, namely the design of I/O-efficient data structures for answering shortest path queries on *planar* directed graphs (digraphs that can be embedded in the plane such that no edges intersect). In particular, we study the space-query trade-off for such structures; using $O(N^2)$ space we can obviously design a structure that can answer a shortest path distance query in $O(1)$ I/Os, simply by storing the shortest paths between every pair of vertices in G. At the other extreme, we can design an $O(N)$ space structure by simply running an SSSP algorithm to answer a query. Since we are interested in massive graphs, we are of course interested in data structures that use close to linear space but answer queries more efficiently that by computing SSSP on-the-fly. In this paper we develop a family of structures with a trade-off between space use and the number of I/Os needed to answer a query. Although this problem has been extensively studied in internal memory [7, 13, 12, 10, 9], this is the first result of its type in external memory.

1.1 I/O-Model and Related Work

We will be working in the standard two-level I/O model [1], 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[1] $M < N$ and $1 \leq B \leq M/2$. 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.

In the I/O-model, the minimal number of I/Os needed to read N input elements (the "linear bound") is obviously $\text{scan}(N) = N/B$. The number of I/Os needed to sort N elements is $\text{sort}(N) = \Theta(\frac{N}{B} \log_{M/B} N/B)$ [1]. For realistic values of N, B, and M, $\text{scan}(N) < \text{sort}(N) \ll N$, and the difference in running time between an algorithm performing N I/Os and one performing $\text{scan}(N)$ or $\text{sort}(N)$ I/Os can be very significant.

On general digraphs the best known algorithm for SSSP, as well the best algorithms for the simpler BFS and DFS problems, use $\Omega(|V|)$ I/Os. More precisely, their I/O-complexity is $O(\min\{(|V| + |E|/B) \cdot \log |V| + \text{sort}(|E|), |V| + \frac{|V|}{M}\frac{|E|}{B}\})$ [8, 11, 17] . However, improved algorithms have been developed for *planar* digraphs [5, 6, 3]. On such graphs, SSSP and BFS can be solved in $O(\text{sort}(N))$ I/Os [5], and DFS in $O(\text{sort}(N) \log N/M)$ I/Os [6]; all these algorithms are based on I/O-efficient reductions [2, 4, 5, 6] and on an $O(\text{sort}(N))$ I/O planar graph separator algorithm [19].

[1] The planar separator algorithm [19] makes the stronger but realistic assumption that $M > B^2 \lg^2 B$; we make this assumption indirectly as we rely on planar separators.

The only known I/O-efficient external data structure for answering shortest path queries is a structure for planar digraphs in [16]. The structure uses $O(N\sqrt{N})$ space and answers shortest path distance queries in $O(\sqrt{N}/B)$ I/Os (and can report the shortest path with additional $O(K/B)$ I/Os, where K is the number of edges on the path). Note that the space-query product is $O(N^2/B)$. The structure in [16] is based on an internal memory data structure obtained independently by Arikati *et al* [7] and Djidjev [12] using planar separators and ideas due to Frederickson [14, 15]. In internal memory, this structure has been generalized to obtain a family of structures, such that a structure using $S \in [N, N^2]$ space can answer shortest path distance queries in $O(N^2/S)$ time [12, 10]. Note that the space-query product is $O(N^2)$. Improved results have been obtained for values of S larger then $N^{4/3}$ [12, 10], as well as for special classes of graphs [13, 9]. Similar space-query results and improvements have not been obtained in external memory; the $O(N\sqrt{N})$ space use of the structure by Hutchinson *et al* [16] probably means that it is mostly of theoretical interest if N is large. Ideas from previous work do not extend in external memory to small space. Finding space-query trade-offs for close to linear S is harder and it is precisely the small values of S that are interesting in external memory.

1.2 Our Results

In this paper we obtain the first space-query trade-offs for external data structures for answering shortest path queries on planar digraphs. Our main result is a family of structures that can answer shortest path distance queries in $O(\frac{N^2}{SB})$ I/Os using $S = \Omega(N^{1+\epsilon})$ (and $S = O(N^2/B)$) space, for any $\epsilon > 0$. Note that, similarly to the internal memory results, the space-query product is $O(N^2/B)$. An S space structure can be constructed in $O(\sqrt{S} \cdot \mathrm{sort}(N))$ I/Os. For values of $S = o(N^{1+\epsilon})$, we show that we can still achieve a trade-off but at the cost of an increased space-query product. More precisely, we show that for any $S \in [N\frac{\log^2 N}{\log\log N}, \frac{N^2}{B}]$, there exists a data structure of size S that can answer distance queries in $O(\frac{N^2}{SB} \cdot \frac{\log N}{\log(S/(N\log N))})$ I/Os. Our structures can be extended to answer shortest path queries with an extra $O(K/B)$ I/Os, where K is the number of edges on the path; for brevity we only consider distance queries.

Our results use ideas similar to the ones in the previously developed internal structures, i.e. planar separators, but with non-trivial external memory modifications to make the structure efficient for small S. In Sec. 2 we review planar separators, and in Sec. 4 we present our new structure. It relies on a structure for computing distances to the boundary of a planar graph presented in Sec. 3.

2 Preliminaries

A $f(N)$-*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 $2N/3$. Lipton and Tarjan [18] showed that any planar

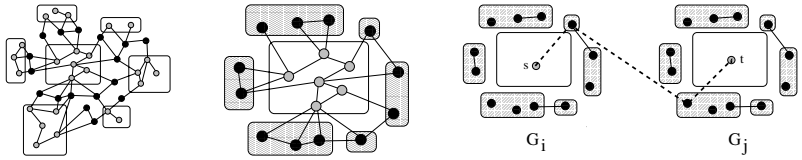

Fig. 1. (a) Partition of G into clusters G_i (boxed) and separators vertices V_S (black). (b) One cluster G_i in the partition and its adjacent boundary sets. (c) The shortest path from s to t is is $\delta(s,t) = \min_{v \in \partial G_i, w \in \partial G_j}\{\delta_{\overline{G_i}}(s,v) + \delta(v,w) + \delta_{\overline{G_j}}(w,t)\}$.

graph has an $O(\sqrt{V})$-separator. Using this result recursively, Frederickson [15] showed that for any parameter $R \leq N$ 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 (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 G_i. These sets are called the *boundary sets* of the partition (Fig. 1(b)). If the graph has bounded degree, which can be ensured for planar graphs using a simple transformation, there exists an R-partition with only $O(N/R)$ boundary sets [15]. It is shown in [19] how to compute such an R-partition in $O(\mathrm{sort}(N))$ I/Os, provided that $M > B^2 \log^2 B$.

All the known internal memory shortest path data structures for planar graphs exploit R-partitions [12, 7, 10]. Consider an R-partition of a planar digraph, and let $\delta_{\overline{G_i}}(s,t)$ denote the length of the shortest path from vertex s to vertex t in $\overline{G_i}$. The shortest path from $s \in G_i$ to $t \in G_j$ must go through the boundaries ∂G_i and ∂G_j of G_i and G_j, and thus we can compute the distance from s to t as $\delta(s,t) = \min_{v \in \partial G_i, w \in \partial G_j}\{\delta_{\overline{G_i}}(s,v) + \delta(v,w) + \delta_{\overline{G_j}}(w,t)\}$ (Fig. 1(c)). The basic idea is to store the distance from a set of vertices of G to the separator vertices V_S in order to be able to efficiently evaluate this formula for given s and t. The size of this set of vertices is a function of the available space S: For small values of S ($S \in [N, N^{3/2}]$) we store only the distances between separator vertices; to answer a query we basically need to solve a SSSP problem in the two clusters $\overline{G_i}$ and $\overline{G_j}$. For large values of S ($S \in [N^{3/2}, N^2]$) we store the distances from all vertices in G to all separator vertices, as well as a shortest path data structure [7, 12] for each cluster; answering a query reduces to using the stored distances if s, t are in different clusters, or querying the cluster, otherwise.

For large values of S ($S \in [N^{2/3}, N^2]$) we can adapt the above strategy to external memory using the known external shortest path data structure [16]; we obtain a structure that answers distance queries in $O(\frac{N^2}{SB})$ I/Os, i.e. has the desired $O(N^2/B)$ space-query product (details in the full version of this

paper). However, for small values of S ($S \in [N, N^{3/2}]$), which are the ones we are normally interested in when handling massive graphs, a similar adaption leads to an $O(\text{sort}(\frac{N^2}{S}))$ query bound, mainly because of the need to solve SSSP problems in two clusters. For small values of S, this is no better than running SSSP from scratch. Note the difference between answering distance queries in internal and external memory: In internal memory small values of S are easy to handle because the clusters in the R-division are small enough for it to be efficient to compute SSSP in linear time in the cluster on-the-fly. In external memory the problem is easy if S is large because we can store enough additional information using S, and becomes harder as S gets smaller. However it is precisely the small values of S that are interesting external memory.

In this paper we show how to use the R-partition in a novel way in order to obtain a small space data structure. One of the main ingredients in our solution is a structure for answering *all-boundary-shortest-path queries*, that is, for finding shortest paths lengths from a vertex s in a cluster G_i to the vertices on the boundary ∂G_i. We describe such a structure below.

3 All-Boundary-Shortest-Path Structure

Assume we are given a cluster H of size N and its boundary ∂H such that $|\partial H| \leq c \cdot \sqrt{N}$, for some constant $c \geq 1$. As usual, we denote $H \cup \partial H$ as $\overline{H}$ and let $\delta_{\overline{H}}(s,t)$ denote the length of the shortest path from s to t in $\overline{H}$. Let $\delta_{\overline{H}}(s, \partial H)$ denote the list of distances from s to the vertices in ∂H, sorted by the id of the vertices in ∂H. Similarly, let $\delta_{\overline{H}}(\partial H, s)$ denote the list of distances from the vertices in ∂H to s, sorted by the id of the vertices in ∂H.

This section describes an I/O-efficient data structure for all-boundary-shortest-path queries, that is, for finding the shortest paths $\delta_{\overline{H}}(s, \partial H)$ and $\delta_{\overline{H}}(\partial H, s)$ between a vertex s in the cluster and the vertices on its boundary. Our structure improves the straightforward $O(\text{sort}(N))$ I/Os bound obtained by running SSSP in $\overline{H}$. More precisely, we prove the following.

Lemma 1. *Given an N-vertex cluster H and its $O(\sqrt{N})$-vertex boundary ∂H we can construct a data structure using $O(N \log N)$ space such that $\delta_{\overline{H}}(s, \partial H)$ or $\delta_{\overline{H}}(\partial H, s)$ can be computed in $O(N/B)$ I/Os for any s in H. The structure can be constructed in $O(\sqrt{N} \cdot \text{sort}(N))$ I/Os.*

Our all-boundary-shortest-path data structure is constructed as follows[2]: we first compute an $N/2$-partition for H, that is, a partition of H using a set V_S of $O(\sqrt{N})$ separator vertices into $O(1)$ clusters, each of which contains at most $N/2$ vertices and is adjacent to at most $\sqrt{N}/2$ separators. As usual define the boundary ∂H_i of a cluster H_i to be the set of vertices in $\partial H \cup V_S$ that are adjacent to vertices in H_i. We then recursively construct an all-boundary-shortest-path data structure for each cluster H_i and its boundary (to do so we first process

[2] We only discuss how to compute $\delta_{\overline{H}}(s, \partial H)$. Computing $\delta_{\overline{H}}(\partial H, s)$ can be done similarly.

each H_i in turn such that its boundary has at most $c \cdot \sqrt{|H_i|}$ vertices; details in the full paper). For each separator or boundary vertex $u \in V_S \cup \partial H$ we compute the shortest paths in $\overline{H}$ from u to all vertices v in ∂H. We store these distances ordered by the vertex id of $v \in \partial H$ and secondarily by vertex id of u; thus the list of distances from ∂H_i to a vertex $v \in \partial H$ can be retrieved by scanning this list in $|V_S \cup \partial H|/B = \sqrt{N}/B$ I/Os.

Query: Consider an all-boundary-shortest-path query $\delta_{\overline{H}}(s, \partial H)$. The interesting case is when $s \in H_i$ (if s is a separator vertex we simply return the list of $O(\sqrt{N})$ pre-computed distances from s to ∂H). Let w be an arbitrary vertex in ∂H; we compute $\delta(s, w)$ as the shortest way to get from s to a boundary vertex v of H_i in $\overline{H_i}$ and from v to w in $\overline{H}$: that is, $\delta(s, w) = \min_{v \in \partial H_i}\{\delta_{\overline{H_i}}(s, v) + \delta(v, w)\}$. It can be shown that $\delta(s, w)$ is indeed the shortest path from s to w in $\overline{H}$. To compute $\delta_{\overline{H}}(s, \partial H)$ we first find the all-boundary-shortest-paths $\delta_{\overline{H_i}}(s, \partial H_i)$ using the recursive data structure for the cluster $\overline{H_i}$ containing s; let L be the list of shortest paths returned, $L = \{\delta_{\overline{H_i}}(s, v) | v \in \partial H_i\}$, sorted by the id of the vertex $v \in \partial H_i$. For every vertex $w \in \partial H$, we compute $\delta(s, w)$ by scanning in parallel the list L and the list of distances from $v \in V_S \cup \partial H$ to w (stored in the structure) and compute the minimum sum $\delta(s, v) + \delta(v, w)$. This takes $O(\sqrt{N}/B)$ I/Os for every $w \in \partial H$, or $O(\sqrt{N} \cdot \sqrt{N}/B) = O(N/B)$ I/Os in total. Thus, the number of I/Os to answer an all-boundary-shortest-path query $\delta_{\overline{H}}(s, \partial H)$ is given by the recurrence $Q(N) = O(N/B) + Q(N/2)$ with solution $Q(N) = O(N/B)$ I/Os.

Space: Storing the shortest paths in $\overline{H}$ from $u \in V_S \cup \partial H$ to all vertices v in ∂H uses $O(\sqrt{N} \cdot \sqrt{N}) = O(N)$ space. Thus the total space is given by $S(N) \leq O(N) + 2S(N/2)$ with solution $S(N) = O(N \log N)$.

Construction: Processing each cluster H_i such that its boundary has at most $c \cdot \sqrt{|H_i|}$ vertices can be done as in the R-partition algorithm [19]; this uses $O(\text{sort}(N))$ I/Os in total for all clusters. Computing the shortest paths in $\overline{H}$ from any vertex $u \in V_S \cup \partial H$ to all vertices v in ∂H can be done in $O(\sqrt{N} \cdot \text{sort}(N))$ I/Os. Thus the total pre-processing time is given by the recurrence $P(N) \leq O(\sqrt{N} \cdot \text{sort}(N)) + 2P(N/2)$ with solution $P(N) = O(\sqrt{N} \cdot \text{sort}(N))$ I/Os. This concludes the proof of Lemma 1.

4 Shortest Path Data Structure

This section describes the main result of this paper, a data structure for shortest path queries. As all related work, our structure is based on R-divisions and pre-computing shortest paths between separator vertices, but it combines these ideas in a novel way that avoids computation of SSSP inside the cluster and obtains an optimal space-time trade-off.

Let R be a parameter $B \leq R \leq N/2$ and G a bounded-degree[3] planar digraph. Our shortest path data structure for G is constructed as follows:

1. Step 1: Compute an R-partition of G.
2. Step 2: Compute and store the distances between the separator vertices V_S, such that the list of distances between a vertex in ∂G_i and the $O(\sqrt{R})$ vertices on the boundary of any cluster ∂G_j can be retrieved sorted by vertex id using $O(\sqrt{R}/B)$ I/Os.
3. Step 3: Recursively construct a shortest path structure for each cluster G_i.
4. Step 4: We do the following for each cluster G_i
   (a) If $M < R \leq N^{2/3}$, we compute and store the distances between all vertices in G_i and vertices in ∂G_i, such that for any $s \in G_i$, $\delta_{\overline{G_i}}(s, \partial G_i)$ and $\delta_{\overline{G_i}}(\partial G_i, s)$ can be retrieved sorted by vertex id using $O(\sqrt{R}/B)$ I/Os.
   (b) If $R > N^{2/3}$, we construct an all-boundary-shortest-path data structure (Lemma 1) for each $\overline{G_i}$.

Below we show how to answer distance queries using the data structure and analyze its space usage and construction time.

Answering distance queries: To find the distance $\delta(s,t)$ between two query vertices s and t we consider the following 4 cases:

(1) If both s and t are separator vertices then we know that $\delta(s,t)$ is stored explicitly (Step 2) and we can thus answer a query in $O(1)$ I/Os.

(2) If s is a separator vertex and t is in cluster G_i (or the symmetrical case) then it can be shown that $\delta(s,t) = \min_{v \in \partial G_i}\{\delta(s,v) + \delta_{\overline{G_i}}(v,t)\}$. To answer the query we first obtain the list of distances from s to ∂G_i (sorted by vertex id) using $O(\sqrt{R}/B)$ I/Os (they are stored explicitly in Step 2). Then we obtain the list $\delta_{\overline{G_i}}(\partial G_i, t)$ of distances from vertices on ∂G_i to t (sorted by vertex id) in $O(R/B)$ I/Os as follows: If $R \leq M$ we load $\overline{G_i}$ in memory using $O(R/B)$ I/Os and compute the distances on the fly. If $M < R \leq N^{2/3}$ the distances $\delta_{\overline{G_i}}(\partial G_i, t)$ are stored explicitly in the data structure (Step 4 (a)) and we can retrieve them in $O(\sqrt{R}/B)$ I/Os; If $R > N^{2/3}$ we obtain $\delta_{\overline{G_i}}(\partial G_i, t)$ from the all-boundary-shortest-path data structure for $\overline{G_i}$ (Step 4(b)) using $O(R/B)$ I/Os (Lemma 1). Since the two obtained lists of distances (from s to ∂G_i and from ∂G_i to t) are both sorted by the vertex id of v we can scan them together to compute the min sum $\delta(s,v) + \delta(v,t)$; thus we answer the query using $O(R/B)$ I/Os in total.

(3) If s is in cluster G_i and t is in a different cluster G_j then it can be shown that $\delta(s,t) = \min_{v \in \partial G_i, w \in \partial G_j}\{\delta_{\overline{G_i}}(s,v) + \delta(v,w) + \delta_{\overline{G_j}}(w,t)\}$. We obtain the lists of distances $\delta_{\overline{G_i}}(s, \partial G_i)$ sorted by vertex id of $v \in \partial G_i$ and $\delta_{\overline{G_j}}(\partial G_j, t)$ sorted by vertex id of $w \in \partial G_j$ using $O(R/B)$ I/Os as in the previous case. We compute $\delta(s,t)$ as follows: scan the list $\delta_{\overline{G_i}}(s, \partial G_i)$ and read the distance from s to the next vertex v on ∂G_i. For each such distance we scan the distances $\delta(v, \partial G_j)$ and $\delta_{\overline{G_j}}(\partial G_j, t)$. Since these lists are sorted by id of vertex in ∂G_j, we

[3] Any graph can easily be transformed into a graph with each vertex having degree at most 3 [15].

can compute $\delta_{\overline{G_i}}(s,v) + \delta(v,w) + \delta_{\overline{G_j}}(w,t)$ for each vertex $w \in \partial G_j$ by scanning the lists in parallel. Thus, for every vertex in ∂G_i we scan two lists of size $\sqrt{R}$ each. Throughout this step we keep a running minimum of the smallest distance encountered so far. This takes in total $\sqrt{R} \cdot \sqrt{R}/B = O(R/B)$ I/Os.

(4) If s and t are in the same cluster G_i then it can be shown that $\delta(s,t) = \min\{\delta_{G_i}(s,t), \min_{v,w \in \partial G_i}\{\delta_{\overline{G_i}}(s,v) + \delta(v,w) + \delta_{\overline{G_i}}(w,t)\}\}$. To answer a query in this case we compute $\delta_{G_i}(s,t)$ using the recursive structure for G_i (Step 3) and the other term in the minimum in $O(R/B)$ I/Os as above. Computing the overall minimum thus takes $O(R/B)$ I/Os. Overall the number of I/Os used to answer a query is given by the recurrence $Q(N) = O(R/B) + Q(R)$ with solution $Q(N) = O(R/B)$.

Construction. An R-partition of G (Step 1) can be computed in $O(\text{sort}(N))$ [19]. The shortest paths in G between the $O(N/\sqrt{R})$ separator vertices (Step 2) can be computed in $O(N/\sqrt{R} \cdot \text{sort}(N))$ I/Os using the $O(\text{sort}(N))$ I/O SSSP algorithm [5] for each separator vertex. Let L be the list containing the shortest paths between the separator vertices: $L = \{\delta(u,v) | u,v \in V_S\}$. We need to store L such that for any separator vertex u, the distances between u and the boundary of any cluster G_j can be retrieved from the list in $\text{scan}(|\partial G_j|) = O(\sqrt{R}/B)$ I/Os. To do so we first tag each separator vertex with the indices of the clusters it is adjacent to. Then by sorting and scanning, we merge this list with L; this gives an augmented list that contains, for every pair of separator vertices (u,v), the distance $\delta(u,v)$ and the indices of the clusters that contain u and v respectively on their boundary; we then scan the augmented list and, for each pair (u,v), for every i such that $u \in \partial G_i$ and for every j such that $v \in \partial G_j$, we output $(u,v,\delta(u,v),i,j)$. Because we assume G to have bounded degree, each separator vertex is adjacent to $O(1)$ clusters; therefore every pair (u,v) appears in the list $O(1)$ times and the size of the list remains $O(|L|) = O(N^2/R)$. Finally, we sort this list by (i,j) and, within the same cluster, by vertex id. Overall we use $O(\text{sort}(N^2/R))$ I/Os. It can be seen that the distances between a vertex $u \in \partial G_i$ and all vertices in ∂G_j are adjacent in the constructed list and can thus be retrieved in $O(\sqrt{R}/B)$ I/Os. Also, the list L_{ij} of $O(R)$ shortest path distances from ∂G_i to ∂G_j can be retrieved in $O(R/B)$ I/Os. Overall the preprocessing required by Step 2 of our data structure uses $O(N/\sqrt{R} \cdot \text{sort}(N) + \text{sort}(N^2/R))$ I/Os.

Finally, the shortest paths between the boundary vertices of a cluster and all vertices in the cluster (Step 4 (a)) can be computed in $O(\sqrt{R} \cdot \text{sort}(R))$ I/Os for each cluster using the $O(\text{sort}(N))$ I/O SSSP algorithm [5] for each boundary vertex. Processing a cluster into an all-boundary-shortest-path data structure (Step 4 (b)) takes $O(\sqrt{R} \cdot \text{sort}(R))$ I/Os by Lemma 1. Thus the preprocessing required by Step 4 of our data structure uses $O(\frac{N}{R} \cdot \sqrt{R} \cdot \text{sort}(R)) = O(N/\sqrt{R} \cdot \text{sort}(R))$ I/Os.

Overall the total pre-processing time $P(N)$ is given by the following recurrence: $P(N) = O(\text{sort}(N) + N/\sqrt{R} \cdot \text{sort}(N) + \text{sort}(N^2/R) + N/\sqrt{R} \cdot \text{sort}(R)) + N/R \cdot P(R) = O(N/\sqrt{R} \cdot \text{sort}(N)) + N/R \cdot P(R)$ with solution $P(N) = O(N/\sqrt{R} \cdot \text{sort}(N))$ I/Os.

Space. Storing the shortest paths between all $O(N/\sqrt{R})$ separator vertices in the partition (Step 2) uses $O(N^2/R)$ space. If $M < R \leq N^{2/3}$ (Step 4(a)) we use $N/R \cdot R\sqrt{R} = O(N\sqrt{R})$ space in total to store the distances between each vertex and all vertices on the boundary of its cluster, which is $O(N^2/R)$. If $R > N^{2/3}$ (Step 4(b)) we use $O(R \log R)$ space on the all-boundary-shortest-path structures (Lemma 1) in each of the N/R clusters, for a total of $O(N \log R)$ space. The total space used by the data structure is thus given by the recurrence $S(N) = O(N^2/R) + O(N \log R) + N/R \cdot S(R)$ with solution $S(N) = O(\max\{N^2/R, N \log R\} \cdot \log_{N/R} N)$.

Overall we have proved the following result:

Theorem 1. *Given a planar digraph G and $R \in [B, N/2]$, a data structure of size $O(\max\{N \log R, N^2/R\} \cdot \log_{N/R} N)$ can be constructed in $O(\frac{N}{\sqrt{R}} \cdot sort(N))$ I/Os such that distance queries can be answered in $O(R/B)$ I/Os.*

Consider the effect of R on the space and query time of the data structure. Note that $\max\{N \log R, N^2/R\}$ is N^2/R for any $R < N/\log N$ and $N \log R$ otherwise. Choosing $R = B$, we obtain a data structure that uses $\Theta(N^2/B)$ space and answers distance queries in $O(1)$ I/Os. This corresponds to building a B-partition and storing APSP between the $N/\sqrt{B}$ separators. As R increases, the space used by the structure decreases and the query time increases. At the other extreme, choosing $R = \frac{N}{\log N}$ we obtain a data structure of size $\Theta(N\frac{\log^2 N}{\log\log N})$ that answers queries in $O(\frac{N}{B \log N})$ I/Os. A simple calculation shows that if $R > \frac{N}{\log N}$ then the space used by the data structure is $N \log R \cdot \log_{N/R} N = \Omega(N\frac{\log^2 N}{\log\log N})$. This means that when increasing R beyond $N/\log N$ both the space of the data structure and the query time go up and we can no longer trade query time for space. Thus, the space used by our structure is lower bounded by $\Omega(N\frac{\log^2 N}{\log\log N})$. One interesting question is whether one can use less then $o(N\frac{\log^2 N}{\log\log N})$ space and answer distance queries faster than $O(\text{sort}(N))$ I/Os.

To express the query time directly in terms of the space S used by the data structure we let R such that $S = O(\frac{N^2}{R} \cdot \log_{N/R} N)$, or $\frac{1}{R} = \frac{1}{N}\frac{S}{N \log N} \log \frac{S}{N \log N}$. Substituting in Theorem 1 above, we obtain:

Lemma 2. *Given a planar digraph G and $S \in [N\frac{\log^2 N}{\log\log N}, \frac{N^2}{B}]$, a data structure of size S can be constructed in $O(\sqrt{\frac{S}{\log N} \cdot \log \frac{S}{N \log N}} \cdot sort(N))$ I/Os such that distance queries can be answered in $O(\frac{N^2}{SB} \cdot \frac{\log N}{\log(S/(N \log N))})$ I/Os*

The bounds in Lemma 2 simplify if S is such that $S/N = \Omega(N^\epsilon)$ for some $\epsilon > 0$. In this case note that $\frac{\log N}{\log(N^\epsilon/\log N)} = \frac{1}{\epsilon} = O(1)$. Thus we obtain our main result stated in Section 1.2:

Theorem 2. *Given a planar digraph G and $S = \Omega(N^{1+\epsilon})$ for some $0 < \epsilon \leq 1$, a data structure of size $O(S)$ can be constructed in $O(\sqrt{S} \cdot sort(N))$ I/Os such that distance queries can be answered in $O(\frac{N^2}{SB})$ I/Os.*

The space-query product of our data structure is $\frac{N^2}{B} \cdot \frac{\log N}{\log(S/(N \log N))}$. For $S = \Omega(N^{1+\epsilon})$ this is $\frac{N^2}{B}$. As S drops below $N^{1+\epsilon}$ the space-query product increases, up to a maximum of $\frac{N^2}{B} \cdot \frac{\log N}{\log \log N}$, when $S = N\frac{\log^2 N}{\log \log N}$.

References

1. A. Aggarwal and J. S. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
2. L. Arge, G. S. Brodal, and L. Toma. On external memory MST, SSSP and multi-way planar graph separation. *Journal of Algorithms*, 53(2):186–2006, 2004.
3. L. Arge, U. Meyer, and L. Toma. External memory algorithms for diameter and all-pairs shortest-paths on sparse graphs. In *Proc. International Colloquium on Automata, Languages, and Programming*, pages 146–157, 2004.
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 N. Zeh. I/O-efficient topological sorting of planar DAGs. In *Proc. ACM Symp. on Parallel Algorithms and Architectures*, 2003.
6. L. Arge and N. Zeh. I/O-efficient strong connectivity and depth-first search for directed planar graphs. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 261–270, 2003.
7. S. Arikati, D. Chen, L. Chew, G. Das, M. Smid, and C. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proc. European Symp. on Algorithms, LNCS 1136*, pages 514–528. Springer, 1996.
8. A. Buchsbaum, M. Goldwasser, S. Venkatasubramanian, and J. Westbrook. On external memory graph traversal. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*, pages 859–860, 2000.
9. S. Chaudhuri and C. Zaroliagis. Shortest path queries in digraphs of small treewidth. In *Proc. International Colloquium on Automata, Languages, and Programming, LNCS 944*, pages 244–255. Springer, 1995.
10. D. Chen and J. Xu. Shortest path queries in planar graphs. In *Proc. ACM Symp. on Theory of Computation*, pages 469–478. ACM Press, 2000.
11. Y. Chiang, M. Goodrich, E. Grove, R. Tamassia, D. Vengroff, and J. S. Vitter. External-memory graph algorithms. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*, pages 139–149, 1995.
12. H. Djidjev. Efficient algorithms for shortest path queries in planar digraphs. In *Proc. Graph-Theoretic Concepts in Comp. Science*, pages 151–165. Springer, 1996.
13. H. Djidjev, G. Pantziou, and C. Zaroliagis. Computing shortest paths and distances in planar graphs. In *Proc. International Colloquium on Automata, Languages, and Programming, LNCS 510*, pages 327–338. Springer, 1991.
14. G. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985.
15. G. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.
16. D. Hutchinson, A. Maheshwari, and N. Zeh. An external-memory data structure for shortest path queries. In *Proc. Annual Combinatorics and Computing Conference, LNCS 1627*, pages 51–60, 1999.

17. 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.
18. R. Lipton and R. Tarjan. A separator theorem for planar graphs. *SIAM Journal of Applied Math.*, 36:177–189, 1979.
19. A. Maheshwari and N. Zeh. I/O-optimal algorithms for planar graphs using separators. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*, pages 372–381, 2002.