A Divide and Conquer approach to Shortest Paths in Planar Layered Digraphs

S. Sairam, Roberto Tamassia, and Jeffrey Scott Vitter

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-92-15
March 1992

A Divide and Conquer approach to Shortest Paths in Planar Layered Digraphs*

S. Sairam, *Roberto Tamassia*, and *Jeffrey Scott Vitter*
(ss@cs.brown.edu, rt@cs.brown.edu, jsv@cs.brown.edu)

Department of Computer Science
Brown University
Providence, R. I. 02912–1910

Abstract

Computing shortest paths in a directed graph has received considerable attention in the sequential RAM model of computation. However, developing a polylog-time parallel algorithm that is close to the sequential optimal in terms of the total work done remains an elusive goal. We present a first step in this direction by showing that for an n-node planar layered digraph with nonnegative edge-weights the shortest path between any two vertices can be computed in $O(\log^3 n)$ time with n processors in a CREW PRAM. A CRCW version of our algorithm runs in time $O(\log^2 n \log\log n)$ and uses $n \log n / \log\log n$ processors. Our results make use of the existence of special kinds of separators in planar layered digraphs, called one-way separators, to implement a divide and conquer solution.

*The authors can be contacted at the Department of Computer Science, Brown University, Providence, RI02912-1910, ph:(401)-863-7646. Support for the first and third author was provided in part by an National Science Foundation Presidential Young Investigator Award CCR–9047466 with matching funds from IBM, by NSF research grant CCR–9007851, by Army Research Office grant DAAL03–91–G–0035, and by the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014–83–K–0146 and ARPA order 6320, amendment 1. Support for the second author was provided in part by by NSF research grant CCR–9007851, by Army Research Office grant DAAL03–91–G–0035, and by the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014–83–K–0146 and ARPA order 6320, amendment 1.

1 Introduction

Computing shortest paths in directed graphs is a fundamental optimization problem with applications to many areas of computer science and operations research [5,14]. Given a digraph G with nonnegative weights on its edges and two vertices s and t of G, the single-pair shortest path problem consists of determining a directed path from s to t with minimum total weight. Among the well known sequential algorithms for this problem is the classical Dijkstra's algorithm [8], based on a dynamic programming approach. Its time complexity is $O((n+m))\log n)$ if elementary data structures are used, and $O(n\log n + m)$ when implemented with Fibonacci heaps [9]. For the important class of acyclic digraphs, a simple variation of Dijkstra's algorithm runs in time $O(n+m)$. Here n and m denote the number of vertices and edges of G, respectively.

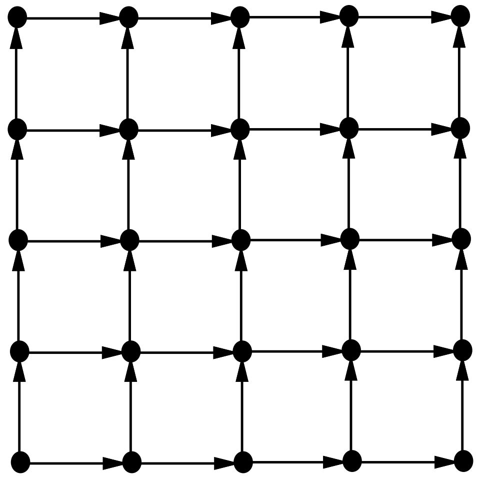

Figure 1: An example of a grid digraph.

Developing a parallel shortest path algorithm that runs in polylogarithmic time with a linear number of processors is an outstanding open problem. Indeed, all the known polylog-time parallel techniques for this problem are based on matrix multiplication [11] and are therefore far from optimal in terms of the total work done, especially when the digraph is sparse. For general digraphs, the best algorithm runs in $O(\log^2 n)$ time with n^3 processors (using the naive algorithm for matrix multiplication) [11]. Recently Alon and Galil [2] have given an algorithm which uses fast matrix multiplication techniques to solve the all-pairs shortest paths problem. The work done by their algorithm is $O((Mn)^{\frac{3+\omega}{2}}\log^3 n)$ if the edges have integral weights which are bounded above by M. Here n^ω denotes the number of processors needed to perform matrix multiplication. However, for calculating single-source shortest paths or for the single-pair shortest path problem even this algorithm is far from optimal. For planar digraphs the number of processors for the single source problem can be reduced to $n^{1.5}/\log n$ [19] while keeping the parallel time down to $O(\log^3 n)$.

To our knowledge, efficient parallel algorithms for computing shortest paths have been devised only for two special classes of digraphs: series-parallel digraphs and grid digraphs. A *series-parallel digraph* [23] is an acyclic digraph with exactly one source and exactly one sink that is recursively constructed by series and parallel compositions. In a series-parallel

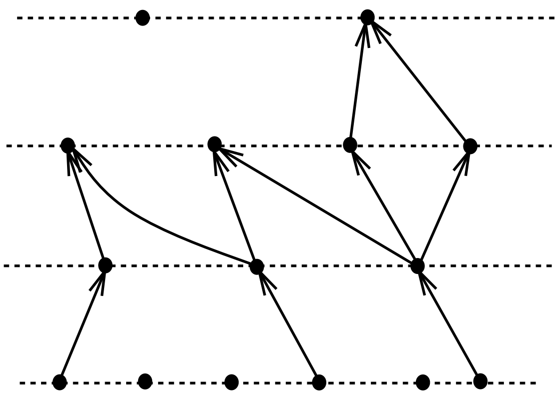

Figure 2: A planar layered digraph.

digraph the weight of a shortest path between the source and the sink is obtained by evaluating an arithmetic expression with operators + (associated with series compositions) and min (associated with parallel compositions), which can be done optimally in $O(\log n)$ time with $n/\log n$ processors [6].

A grid digraph has the vertices arranged in a rectangular grid and the edges directed from left to right and from bottom to top (an example is shown in Figure 1). In such a digraph the shortest path between any two vertices can be determined in $O(\log^2 n)$ time with $O(n)$ processors [3]. The shortest path problem on grid digraphs has applications in text processing, biological research, tomography and medical diagnosis.

In this work we consider a class of digraphs that extends grid digraphs, namely, *planar layered digraphs*. In a planar layered digraph the vertices are arranged along parallel lines, called layers, and edges connect vertices of consecutive layers and do not intersect. We present an efficient parallel algorithm for the shortest path problem in such digraphs that runs in $O(\log^3 n)$ time with n processors. The algorithm uses a divide and conquer approach and is based on the novel idea of a *one-way separator*, which has the property that any directed path can cross it only once. We are currently investigating how to extend our result to the class of planar acyclic digraphs.

The rest of this paper is organized as follows. In Section 2 we formally define planar layered digraphs and introduce the notion of one-way separators. In Section 3 we prove the existence of a one-way separator for planar layered digraphs and show how to use such a separator to design an efficient divide-and-conquer shortest path algorithm. In Section 4 we discuss how our results relate to the previous one for grid digraphs. Finally, in Section 5 we present our conclusions and comment on the open problem of finding shortest paths in planar st-graphs.

2 Preliminaries

Let $G = (V, E)$ be a directed acyclic graph with n vertices and m edges, we say that G is a layered digraph if V is partitioned into k layers $l_1, \ldots, l_k$ such that all edges of G are between consecutive layers. Given k parallel lines consecutively numbered in the plane, a *k-line embedding* of G is a drawing such that

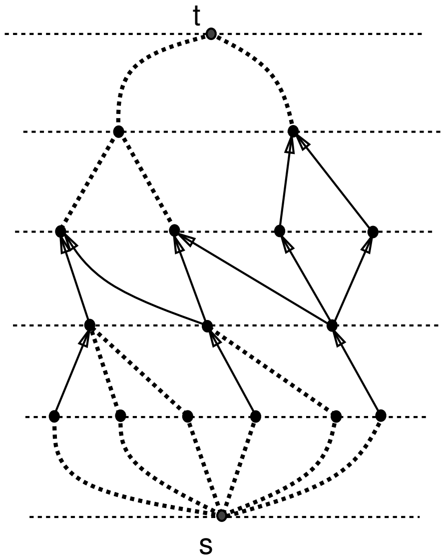

Figure 3: Adding extra edges to create a planar layered st-graph.

- All vertices of layer l_i are drawn on line i;
- Edges are drawn as straight lines

A *planar k-line embedding* is a k-line embedding without crossings. G is a planar layered digraph if it admits a planar k-line embedding. Figure 2 gives an example of a layered graph with a planar 5-line embedding. Layered graphs have been studied under the name of *proper hierarchies* in [25]. Di Battista and Nardelli [7] give efficient algorithms to test if a layered digraph with only one source is planar. Recently Kosaraju [13] has developed an efficient parallel algorithm to evaluate planar layered circuits.

It is easy to see that, by adding edges any planar layered digraph G can be transformed into a planar layered digraph with exactly one source and one sink without affecting the planarity and without altering any of the shortest paths in the original graph. This can be accomplished by introducing edges of infinite weight incident on the sources and sinks, and introducing two new nodes s and t which are respectively source and sink of the new graph (see Figure 3). This transformation can be done in $O(\log n)$ with n processors using techniques in [12]. A planar layered digraph with exactly one source s (on the first layer) and one sink t (on the last layer) is called a planar layered st-graph. In the rest of the paper we solve the problem of finding the shortest path between the source and the sink of a planar layered st-graph. To find the shortest path between any two vertices u and v in G we perform a preprocessing step to remove all the vertices which are not on any path between u and v. A brief outline of the algorithm follows:

1. Let T_u be the set of vertices that are on some path from u to t in G.
2. Let S_v be the set of vertices that are on some path from s to v in G.
3. Discard all the vertices in G that are not in T_u or S_v.

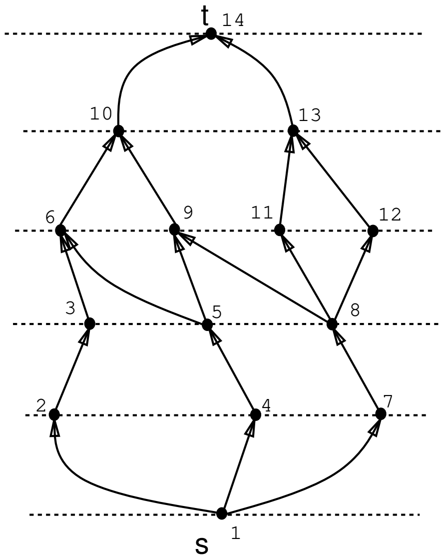

Figure 4: Left ordering of a planar st-graph.

Theorem 1 *The algorithm outlined above can be implemented in $O(\log n)$ time with $n/\log n$ processors in an EREW PRAM.*

Proof: All the steps can be done in $O(\log n)$ time using techniques in [22]. □

A planar layered st-graph is a special case of a planar st-graph which is defined as a planar acyclic digraph with exactly one source, s, and exactly one sink, t, embedded in the plane so that s and t are on the boundary of the external face. These graphs were first introduced in the planarity testing algorithm of Lempel *et al.* [15].

We now define the concept of a *left* ordering of the vertices in a planar st-graph, which will prove useful in our algorithm. This ordering was introduced by Tamassia and Preparata [21]. We do this by making use of the dual graph of G (labeled G^*) defined as follows:

1. Every internal face f in G corresponds to a vertex in G^*.
2. The dual edge e^* of an edge e is directed from the face to the left of e to the face to the right of e.
3. The external face of G is dualized to two vertices of G^*, denoted s^* and t^*, which are incident with the duals of the edges on the leftmost and rightmost paths from s to t, respectively.

Definition 1 For every $x \in V$ we denote by $left(x)$ and $right(x)$ the two faces that separate the incoming and outgoing edges of a vertex $x \neq s, t$. For $x = s$ or $x = t$, we conventionally define $left(x) = s^*$ and $right(x) = t^*$.

Definition 2 We say that x is *below* y, denoted $x \uparrow y$, if there is a path in G from x to y. Also, we say that x is *to the left of* y, denoted $x \rightarrow y$, if there is a path in G^* from $right(x)$ to $left(y)$.

Definition 3 The left ordering (denoted by $<_l$) is defined on the basis of the relations "left" and "below". A vertex x occurs before another vertex y in the ordering if either $x \rightarrow y$ or $x \uparrow Y$.

A planar st-graph with its vertices numbered according to the *left ordering* is shown in Figure 4. Tamassia and Vitter [22] give optimal EREW algorithms to construct the left ordering of a planar st-graph.

Definition 4 Given a graph $G = (V, E)$ with n vertices we call a set $X \subset V$ an $f(n)$-separator that δ-splits G if $|X| \leq f(n)$ and the vertices in $V - X$ can be partitioned into sets $\{A_1, \ldots A_k\}$ such that there are no edges between any two sets A_i and A_j and for all A_i we have $|A_i| \leq \delta n$.

Lipton and Tarjan [16] proved that any planar graph with n vertices has a $\sqrt{8n}$-separator that 2/3-splits. This result and other extensions to it have paved the way to divide and conquer solutions for many problems in planar graphs. A parallel algorithm for finding a cycle separator for biconnected graphs was given by Miller [17] which uses n processors if the breadth first search tree of the graph is already known, an improved version of the algorithm was given by Gazit and Miller [10], which uses randomization to find a cycle separator with $n^{1+\epsilon}$ processors. Randomized parallel algorithms to find small separators for more general undirected graphs are given in [18]. Recently Shannon and Wan [20] have given the first linear-processor deterministic algorithm to find $O(\sqrt{n})$ separators in planar graphs. However, these separators seem unsuitable for use in solving problems on directed planar graphs because they do not take into account the direction of edges, while separating the graph. For example when considering the problem of determining the shortest path from s to t, the existence of a small separator which separates the underlying undirected graph does not guarantee an efficient recursive solution. We get into trouble because even though we can use the separator to divide the original problem into two roughly equal subproblems, the time taken to patch up the two recursive solutions can be large, as the shortest path from s to t may cross the separator many times. For a clean divide and conquer approach we use the following special kind of separator:

Definition 5 Let X be a separator of a digraph $G = (V, E)$ that divides $V - X$ into sets $A_1, \ldots A_k$, and let p be a simple directed path in G. We say that p crosses X r times if there are disjoint subpaths $p_1, \ldots, p_r$ of p such that the end points of each p_i are in different sets A_j and A_l. We call X a *one-way separator* if any directed path p between two vertices in G crosses X at most once.

Grid digraphs for instance have one-way separators of size $\sqrt{n}$ that 1/2-split the graph. In fact the shortest path algorithm in [3] uses such separators to construct a divide and conquer solution. In this paper our main result is that planar layered digraphs also admit small one-way separators which we use to construct a recursive solution to the shortest path problem.

3 Shortest paths in a planar layered st-graph

Let $G = (V, E)$ be a planar layered st-graph with source s and sink t. In this section we show how to determine the shortest path from s to t by a divide and conquer approach. The

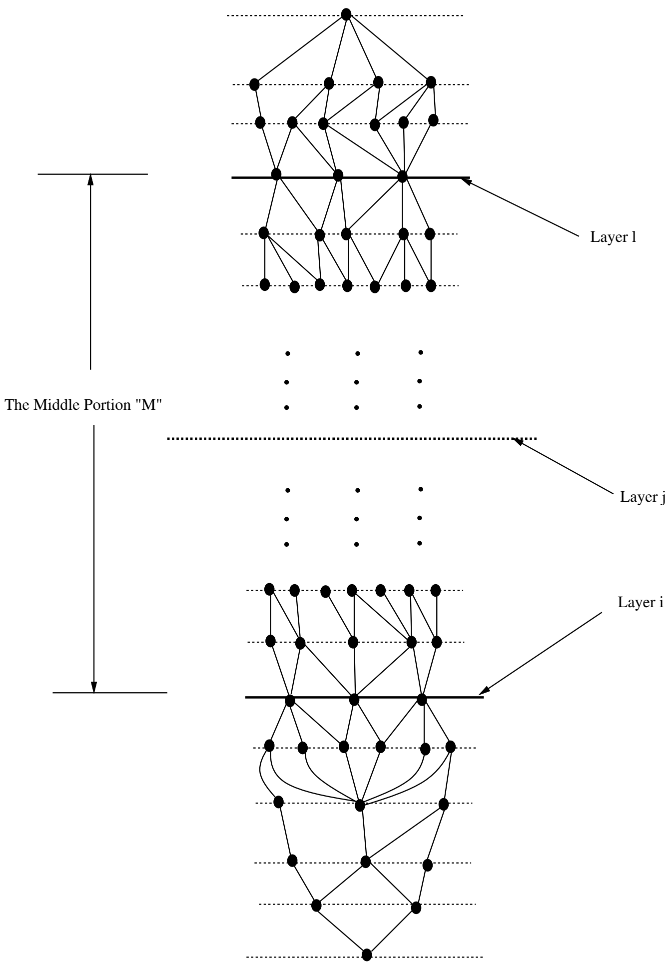

Figure 5: Slicing the graph into three parts.

crux of the algorithm lies in finding a one-way separator. We find a one-way separator X of size $\sqrt{n}$ that 2/3-splits G into at most four parts A, B, C, and D. We use this separator to solve the shortest path problem for the graph G by recursively solving the subproblems in the four pieces and patching the solutions along X. To show the existence of such a separator we need the following lemma, which follows directly from the arguments in [16]:

Lemma 1 *Let G be any n-vertex planar layered st-graph containing layers 1 through k. Let S_i denote the set of vertices in the ith layer, and let n_i denote the size of the set S_i. Also let $k+1$ be an additional layer containing no vertices. Given any layer j there exists a layer $i \leq j$ such that $n_i + 2(j-i) \leq 2\sqrt{t_j}$, where t_j denotes the number of vertices in layers 1 through j. Similarly there exists a layer $l \geq j$ such that $n_l + 2(l-j-1) \leq 2\sqrt{n-t_j}$. In particular we let i (respectively l) be the layer which minimizes the function $n_i + 2(j-i) \leq 2\sqrt{t_j}$ (respectively $n_l + 2(l-j-1) \leq 2\sqrt{n-t_j}$).*

Theorem 2 *Given an n-vertex planar layered st-graph G containing layers 1 through k, there exists a one-way separator consisting of two layers and possibly a path that splits 2/3-splits G into at most four pieces.*

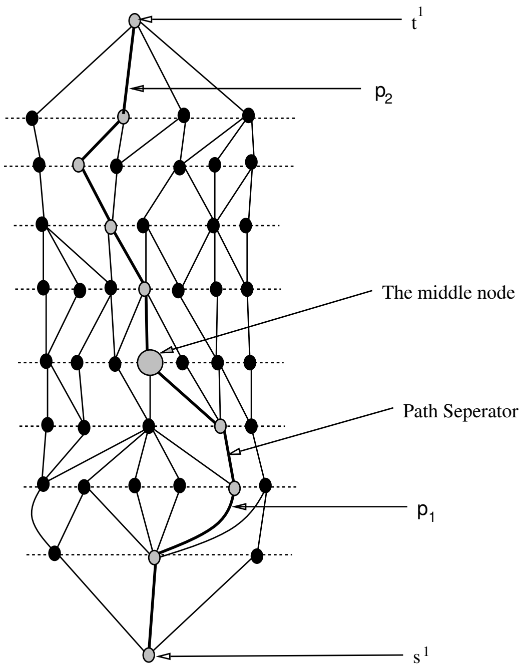

Figure 6: Finding a path separator in the middle graph.

Proof: Given G, let j be the smallest layer such that the layers 1 through j have at least $n/2$ vertices. Let i and l be layers as in Lemma 1. Let A be the part of G above l, let B the part below i, and let M be the remaining part in the middle (as shown in Figure 5). By construction A and B each have less than $2n/3$ vertices. If the number of vertices in M is less than $2n/3$, we label M as C and we are done. Otherwise we form a new planar layered st-graph G' from the middle part M and two new vertices s' and t', adding edges from all the vertices in layer l to t' and from s' to all the vertices in layer i (see Figure 6). Let m be the middle vertex in the left ordering L_1 of G', and let p be a path constructed by taking the leftmost outgoing edge from m until reaching t', and the rightmost incoming edge from m until reaching s' (see Figure 6). The path p splits G' into two equal parts C and D.

We now claim that the separator X consisting of S_i, S_l, and p forms a "one-way" separator, which 2/3-splits G such that $n_i + n_l + |p| \leq 2\sqrt{2n}$. It is easy to see that none of the pieces have more than $2n/3$ vertices. By construction $|p| \leq l - j + 1$, therefore, $n_i + n_l + |p| \leq n_i + n_l + 2(j-i) + 2(l-j-1) \leq 2(\sqrt{t_j} + \sqrt{n-t_j}\,) \leq 2(\sqrt{n/2} + \sqrt{n/2}\,) = 2\sqrt{2n}$. To prove that X forms a one-way separator all we need to show is that no path crosses X more than once. By the definition of planar layered digraphs no path in G intersects with vertices in S_i or S_l more than once (see Figure 5). To look at the interactions of paths in G with the path p, we look at the two sub paths p_1 and p_2 of p, where p_1 is the portion below the middle vertex and p_2 is the portion above. Let C be the subgraph to the left of p and D the subgraph to the right of p. By construction we can see that no path can cross from D to C since no path can enter p_1 from the right, and no path can leave p_2 from the left (see Figures 6 and 7). Therefore, X is a one-way separator. □

We now show how to obtain the separator described in Theorem 2 for a planar layered st-graph G quickly in parallel. A brief sketch of the algorithm is outlined below.

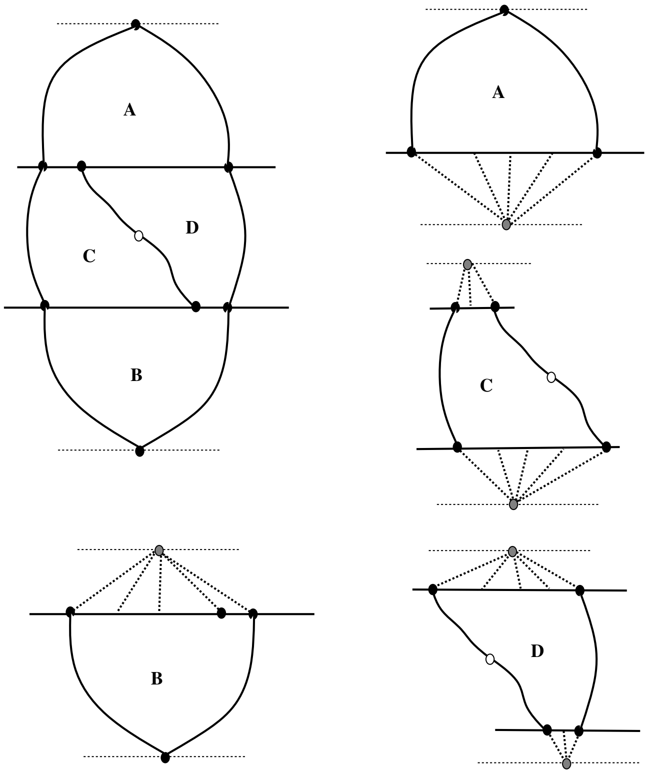

Figure 7: A schematic view of the four pieces obtained by using the separator.

1. For every vertex n determine the level in which it is located.
2. Determine the number of vertices in level i for all i.
3. Determine the levels i, j, and l.
4. Construct the middle graph M and determine its weight. If the weight is less than $2n/3$ we are done.
5. Construct the left ordering of M, and find the middle vertex x.
6. Construct the "left-most outgoing and right-most incoming" separator starting (as shown in Theorem 2) from x.

Theorem 3 *Given an n-vertex planar layered st-graph G the algorithm outlined above constructs the one-way separator of Theorem 2 and can be implemented in time $O(\log n)$ using $n/\log n$ processors on an EREW PRAM.*

Proof: Steps 1 through 4 can be done using standard techniques (see [12]). Steps 5 and 6 can be completed in $O(\log n)$ time using techniques given in [22]. □

We now use X to construct a divide and conquer solution for the shortest path problem. The basic idea is to solve the four all-pairs subproblems (among the boundary vertices) in the pieces A, B, C, and D and patch up the solutions along the separator X efficiently to obtain the shortest path form s to t. We construct planar layered st-graphs from the pieces A,

B, C, and D by introducing new source sink vertices and adding edges from the top and the bottom layers, as shown in Figure 7. It is easy to see that we introduce only $o(n)$ new vertices in this fashion. We still have to show that the number of all pairs subproblems at any stage is not too much to handle with just n processors since we are aiming for a linear processor solution. We use the following lemma which follows from the arguments in [16] to show that the total number of source-sink pairs at any level in the recursion is $O(n \log n)$.

Lemma 2 *Let G be an n-vertex planar layered st-graph, and let $0 \leq \epsilon \leq 1$. If the separator X defined in Theorem 2 is used to split the graph G recursively until we have no piece with more than ϵn vertices, then the number of boundary vertices (vertices which are a part of some separator in the recursive subdivision) is $O(\sqrt{n/\epsilon})$.*

Before counting the number of source-sink pairs at some level in the recursion let us analyze levels more precisely. Every split in our recursion uses a separator X to divide the parent graph P into at most four pieces, each of size at most $2k/3$, where P has k vertices. Any split that divides the graph into more than two pieces can be considered as a sequence of two splits each dividing a parent graph into two pieces. In this manner we build a recursive division tree (RDT), wherein the root corresponds to the graph G, and the leaves to graphs of size at most 2. Each internal vertex has two children, which correspond to the two pieces resulting from a split by a separator (as constructed in Theorem 2). The ith level in the tree consists of all the pieces that are at a distance i from the root. We now claim that at each such level the number of new source-sink pairs introduced is linear.

Lemma 3 *The number of new source-sink subproblems introduced in any level of the recursive division tree is $O(n)$.*

Proof : By induction. The base case $i = 2$ is certainly true, since the separator X we get from Theorem 2 is of size at most $2\sqrt{2n}$, and therefore, the total number of source-sink subproblems introduced in the first level of recursion is at most $8n$. Let us consider some level i in the RDT, such that the induction hypothesis holds for all levels up to $i-1$. We now prove that only $O(n)$ new source-sink pairs are introduced in constructing level i. Let $\epsilon = (2/3)^{(i-1)/2}$, from the definition of RDT, we know that the size of any piece in level $i-1$ is at most ϵn. Therefore, by Lemma 2 the number of boundary vertices is $O(\sqrt{n/\epsilon})$. Let B be the set of boundary vertices in level $i-1$. To construct the next level of the RDT we use separators $X_1, X_2, \ldots, X_l$, each of size at most $2\sqrt{2\epsilon n}$, to divide the pieces in level $i-1$. Any new source-sink pair in level i will involve some separator vertex $a \in X_j$, for $1 \leq j \leq l$ and a boundary vertex $b \in B$. Hence, total number of source sink pairs introduced is at most $2\sqrt{2\epsilon n}\, O(\sqrt{n/\epsilon}) = O(n)$. □

From Lemma 3 we know that the total number of source-sink subproblems in any level is $O(n \log n)$, since there are $O(\log n)$ levels and only $O(n)$ new subproblems are introduced in any level. We use the following strategy to solve the shortest path problem:

1. Recursively subdivide the graph until there are only 2 vertices in any piece.

2. Solve the problem in a bottom up manner, by dynamically assigning processors to each piece depending on the number of source-sink pairs in that piece.

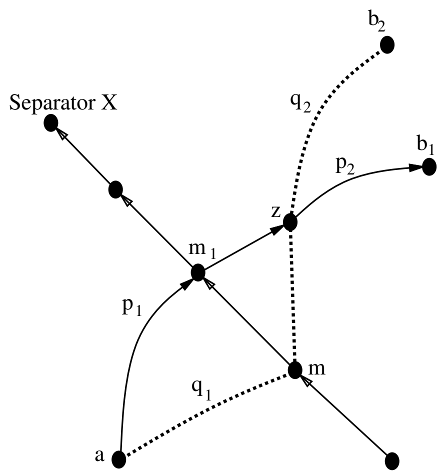

Figure 8: Shortest paths from a are non crossing.

In the remaining part we show how to patch up the results of any two pieces P_1 and P_2 to obtain the shortest path information of the parent graph P at all levels of recursion. To patch up the pieces P_1 and P_2 we need to update the shortest path information along the common boundary. Let us suppose without loss of generality that the source set S is in P_1 and the sink set T is in P_2, and the common boundary between them is X. Let n_s, n_t, n_x be the number vertices in S, T, and X respectively. We know from Theorem 2 that X is either a layer of vertices or a special "right-most-edge-in, left-most-edge-out" path constructed from a middle vertex m. We have already solved the subproblems associated with pieces P_1 and P_2, therefore, we know the all-pairs shortest paths from vertices in S to those in X, and from vertices in X to those in T. We now have to construct the all-pairs shortest paths from vertices in S to those in T. On the face of it this seems an easy enough job, since for each pair of vertices (a, b) for which $a \in S, b \in T$, all we have to do is check for every vertex in X whether it is in the shortest path from a to b or not. This can be done in constant time with n_x processors in a CRCW PRAM. However, a little thought reveals that such a naive approach would require too many processors. Assume without loss of generality that $n_s \leq n_t$. We now use an idea from [3] to show that the all-pairs shortest paths from vertices in S to vertices in T can be computed in polylogarithmic time with $n_s n_t + n_s n_x$ processors. Later we reduce the processors count to $n_s(n_t + n_x)/\log n$). Since $n_s n_t + n_s n_x$ is $O(n \log n)$ by Lemma 3, we obtain a linear processor algorithm. We make use of the following definition and lemma.

Definition 6 Let a and b be two vertices separated by a one-way separator $X \subset V$ in the graph G. We denote by $l(a, b)$ the lowest point in the left ordering of the vertices in X in the graph G such that a shortest path from a to b in G passes through it.

Lemma 4 *Let a be a vertex in S and let b_1 and b_2 be vertices in T. Let $m_1 \in X = l(a, b_1)$ in the parent graph P. We claim that if b_2 occurs after b_1 in the left ordering of the parent graph P then $l(a, b_2)$ does not occur before m_1 in the left ordering.*

Proof: We give the proof for the case when X is a path; the proof when X is a layer is similar. The proof is by contradiction. Let $l(a, b_2) = m$ occur before m_1 in the left ordering of P (see Figure 8). By the construction of the separator we know that both b_1 and b_2 are on the boundaries of the parent graph P (either on the right boundary or on the layer just below the sink as in Figure 7); therefore, the paths from a to b_1 and b_2 must cross each other at some point. Let the point of crossing be z. Consider the two pieces of each path; let the subpath from a to z be p_1, and the one from z to b_1 be p_2. Similarly let the two parts of the path from a to b_2 be q_1 and q_2. Without loss of generality assume that the weight of p_1 is less than that of q_1. We now have a contradiction because the path composed of p_1 and q_2 has weight less than the one made up of q_1 and q_2. □

A similar lemma can be proved about paths from two source vertices to a single sink vertex. We can now use Lemma 4 to do the patching up across the boundary X with $n_s n_t + n_s n_x$ processors. For every vertex $a \in S$ we use $n_t + n_x$ processors to determine simultaneously all the shortest paths p_b^a from a to each $b \in T$.

Let us denote the problem of patching across the separator for a given source vertex a as $P(a, X, T)$, where we are given the shortest paths from a to every vertex in X and between all pairs of vertices (x, b) where x is in X and b is in T, and we need to determine the shortest paths from a to every vertex $b \in T$. The algorithm to solve $P(a, X, T)$ is as follows:

1. Let b_m be the middle vertex in the left ordering of T.
2. Determine the vertex x_m such that $x_m = l(a, b_m)$.
3. If $n_x = 1$ use n_t processors to determine for all $b \in T$ the shortest path from a to b. Otherwise go to Step 4.
4. Divide T into two sets T_1 and T_2 such that $T_1 = \{b \in T \mid b < b_m\}$ and $T_2 = T - T_1$. Similarly divide X into two corresponding sets X_1 and X_2 such that $X_1 = \{x \in X \mid x < x_m\}$ and $X_2 = X - X_1$.
5. Recursively solve the two subproblems $P(a, X_1, T_1)$ and $P(a, X_2, T_2)$.
6. For all vertices $b \in T_1$ compare the shortest path from a to b through X_1, with the one through x_m, and pick the smaller one.

Theorem 4 *Starting with an n-vertex planar layered st-graph, it is possible at any stage of recursion to obtain the shortest path results of the parent graph P from those of its children P_1 and P_2 by running n_s copies of the algorithm outlined above. Moreover this can be done in time $O(\log^2 n)$ in a CREW PRAM, using $(n_s n_t + n_s n_x)/\log n + n_p$ processors, where n_p is the number of vertices in the parent graph P.*

Proof: Steps 1 and 2 are obviously correct. To prove the correctness of Step 4 we make use of Lemma 4. We know from Lemma 4 that for any $b \in T_2$ the shortest path from a to b does not use any vertex in X_1, and for all $b \in T_1$ the shortest path form a to b either passes through X_1 or through x_m. We can therefore recursively divide the problem into two subproblems $P(a, X_1, T_1)$ and $P(a, X_2, T_2)$, and do the correction in Step 6 to determine the shortest paths to vertices in T_1. To determine the processor and time complexity we note that the left ordering of the any planar st-graph with n vertices can be determined in $O(\log n)$ time with n processors [22]. Thus we can determine the orderings of S and T in $O(\log n_p)$ time with n_p processors. The time complexity of the various steps are as follows:

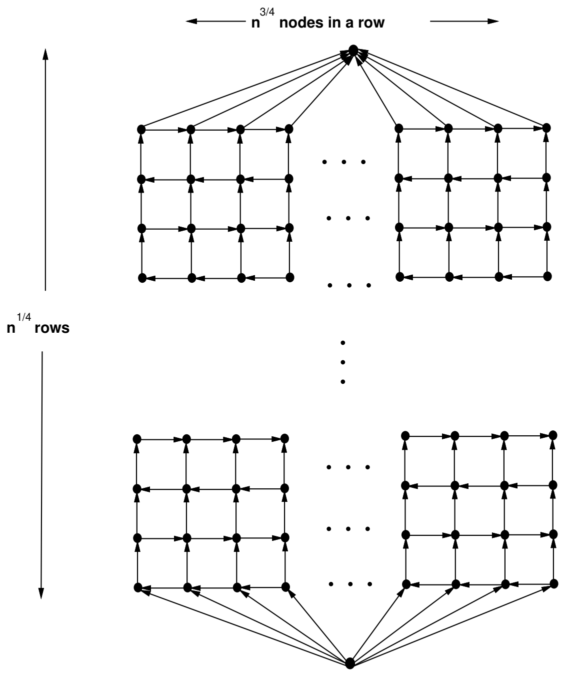

Figure 9: A bad planar st-graph.

1. Step 1 can be performed in $O(\log n_t + \log n)$ time with $n_t / \log n$ processors in an EREW PRAM.

2. Step 2 can be performed in time $O(\log n_x + \log n)$ with $n_x / \log n$ processors in an EREW PRAM.

3. Step 3 takes $O(\log n)$ time with $n_t / \log n$ processors.

4. Step 4 can be done in $O(\log n_t + \log n_x + \log n)$ time with $(n_t + n_x) / \log n$ processors.

5. The depth of recursion is $O(\log n_t)$, since the size of the sink set T goes down by a factor of 2 every time step 4 is executed. Therefore, the total time taken for the patch up is $O(\log n_p + \log^2 n_t + \log n_x + \log n)) = O(\log^2 n)$.

6. Step 6 can be done in constant time with n_t processors in a CREW PRAM.

   To see the processor complexity of the recursive subproblems we note that the two subproblems are solved in parallel by partitioning the processors between them. Subproblem $P(a, X_1, T_1)$ gets $(|X_1| + |T_1|) / \log n$ processors while subproblem $P(a, X_2, T_2)$ gets $(|X_2| + |T_2|) / \log n$ processors. The total number of processors needed is $n_s(n_x + n_t) / \log n + n_p$.

□

Theorem 5 *Given a n-vertex planar layered st-graph G with nonnegative edge weights, we can determine the shortest path from s to t in time $O(\log^3 n)$ with n processors in a CREW PRAM.*

Proof: The proof immediately follows from Lemma 3 and Theorems 3 and 4. □

4 Remarks

Consider at any stage of the computation the array A, where $A[i,k,j]$ is the value of the shortest path from vertex $i \in S$ to vertex $j \in T$ that is constrained to pass through the vertex $k \in X$. Let $\Theta_A(i,j)$ be the smallest index k that minimizes $A[i,k,j]$. We say that A is totally monotone if Θ_A satisfies the following properties:

$$\Theta_A(i,j) \leq \Theta_A(i+1,j);$$

$$\Theta_A(i,j) \leq \Theta_A(i,j+1).$$

Furthermore every submatrix A' also satisfies these properties. It can be shown that at every stage of the computation the matrix A constructed as above is totally monotone. It is not hard to see that to patch up two subproblems P_1 and P_2 we only need to determine the value of $\Theta_A(i,j)$ for all i,j. The problem of determining the value of Θ was formalized by Agarwal and Park as the *Tube minima* problem and has many applications in computational geometry [1]. Atallah [4] has recently given an efficient parallel algorithm for determining the tube minima of an $n \times n \times n$ totally monotone matrix using $O(n^2/\log\log n)$ processors which runs in time $O(\log\log n)$ in a CRCW PRAM. In the case of grid graphs where each intermediate array A is an $\sqrt{n} \times \sqrt{n} \times \sqrt{n}$ array this result can be used to determine shortest paths with n processors in time $O(\log n \log\log n)$ in a CRCW PRAM. However, there seems to be no obvious generalization of Atallah's algorithm where the number of sources and sinks at any intermediate stage are not necessarily equal. One of the open problems is to develop a tube minima algorithm for an $n_1 \times n_2 \times n_3$ totally monotone matrix (where $n_1 < n_3$) that does $O(n_1n_2 + n_1n_3)$ work and runs in time $O(\log\log(n_1+n_2+n_3))$. Such an algorithm would also give the fastest possible algorithm for the row minima problem (which is the same as the tube minima problem but applied to a two-dimensional array), in light of the $\Omega(\log\log n)$ lower bound by Valiant [24] for finding the minimum of n integers with $O(n)$ work by using only comparisons.

5 Conclusions

We have given a linear processor CREW algorithm for determining the shortest paths in a planar layered digraph which runs in time $O(\log^2 n)$. Our motivation for considering layered graphs was to see if they can be used to solve the shortest path problem for planar st-graphs however since they extend the class of grid digraphs they may have other applications. As far as planar st-graphs are concerned it seems unlikely that a divide and conquer approach would give a linear processor solution for the shortest path problem. As evidence we present the graph in Figure 9 that does not have a one-way separator of size $O(\sqrt{n})$. This suggests that we may have to look for iterative solutions when considering the class of planar st-graphs.

References

[1] A. Agarwal and J. Park, "Notes on Searching Multidimensional Monotone Arrays," *Proc. ACM Symp. on Foundations of Computer Science* (1988), 496–512.

[2] N. Alon and Z. Galil, "On the Exponent of the All Pairs Shortest Path Problem," *Proceedings of the 32nd Symposium on Foundations of Computer Science* (1991), 569–575.

[3] A. Apostolico, M. J. Atallah, L. Larmore, and H. S. Mcfaddin, "Efficient Parallel algorithms for String Editing and Related Problems ," *SIAM J. Computing* 19 (1990), 968–988.

[4] M.J. Atallah, "A Faster Parallel Algorithm for a Matrix Searching Problem," *Proc. 2nd Scandinavian Workshop on Algorithm Theory* (1990), 193–200.

[5] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics* 16 (1958), 87–90.

[6] R. Cole and U. Vishkin, "Optimal parallel algorithms for expression tree evaluation and list ranking ," *Procedings of the third Agean Workshop on Computing:* (1988), 91–100.

[7] G. Di Battista and E. Nardelli, "An Algorithm for Testing Planarity of Hierarchical Graphs," *Proc. In. Workshop WG8́6, Bernierd, June 1986.* (1987), 277–289.

[8] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik* 1 (1959), 269–271.

[9] M.L. Fredman and R.E. Tarjan, "Fibonacci Heaps and their Uses in Improved Network Optimization Algorithms," *J. ACM* 34 (1987), 596–615.

[10] H. Gazit and G. Miller, "A parallel algorithm for finding a seperator in planar graphs," *Proc. FOCS,1987* (1987), 238–248.

[11] M. Gondran and M. Minoux, in *Graphs and Algorithms*, Wily InterScience New York, 1984.

[12] R.M. Karp and V. Ramachandran, "A Survey of Parallel Algorithms for Shared Memory Machines," in *Handbook of Theoretical Computer Science*, North Holland, 1990.

[13] S. Kosaraju, Personal Communication, 1991.

[14] E. Lawler, *Combinatorial Optimization*, Holt, Rinehart and Winston, 1976.

[15] A. Lempel, S. Even, and I. Cederbaum, "An Algorithm for Planarity Testing of Graphs," *Theory of Graphs, Int. Symposium* (1966), 215–232.

[16] R. J. Lipton and R. E. Tarjan, "Applications of a Planar Separator Theorem," *SIAM J. Computing* 9 (1980), 615–627.

[17] G. Miller, "Finding small simple cycle seperators for 2-connected planar graphs," *Journal of Computer and System Sciences* 32 (1986), 265–279.

[18] G. Miller and W. Thurston, "Seperators in Two and Three Dimensions," *Proc. STOC, 1990* (1990), 300–309.

[19] V. Pan and J. Reif, "Extension of the Parallel Nested Dissection Algorithm to the Path Algebra Problems," Aiken Computation Lab, Harvard University, Technical Report TR-15-85, 1985.

[20] G. Shannon and F. Wan, "Subdividing Planar Graphs in Prallel," Department of Computer Science Indiana University, Bloomington, Technical Report, 1991.

[21] R. Tamassia and F.P. Preparata, "Dynamic Maintenance of Planar Digraphs, with Applications," *Algorithmica* 5 (1990), 509–527.

[22] R. Tamassia and J.S. Vitter, "Parallel Transitive Closure and Point Location in Planar Structures," *SIAM J. Computing* 20 (1991 (to appear)).

[23] J. Valdes, R.E. Tarjan, and E.L. Lawler, "The Recognition of Series Parallel Digraphs," *SIAM J. on Computing* 11 (1982), 298–313.

[24] L. Valiant, "Parallelism in Comparison Problems," *SIAM Journal of Computing* 4 (1975), 348–355.

[25] S. Whitesides, "Forms of hierarchy: A selected bibliography," *Gen. Syst.* 14 (1969), 3–15.