Algorithmica (1995) 14: 322–339

Algorithmica

An Efficient Parallel Algorithm for Shortest Paths in Planar Layered Digraphs[1]

S. Subramanian,[2] R. Tamassia,[2] and J. S. Vitter[3]

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 giving efficient parallel algorithms for shortest paths in planar layered digraphs.

We show that these graphs admit special kinds of separators called *one-way* separators which allow the paths in the graph to cross it only once. We use these separators to give divide-and-conquer solutions to the problem of finding the shortest paths between any two vertices. We first give a simple algorithm that works in the CREW model and computes the shortest path between any two vertices in an n-node planar layered digraph in time $O(\log^2 n)$ using $n/\log n$ processors. We then use results of Aggarwal and Park [1] and Atallah [4] to improve the time bound to $O(\log^2 n)$ in the CREW model and $O(\log n \log\log n)$ in the CREW model. The processor bounds still remain as $n/\log n$ for the CREW model and $n/\log\log n$ for the CRCW model.

Key Words. Parallel algorithms, Shortest paths, Planar separators.

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], [20]. 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 [10], 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 [12]. For the important class of acyclic

[1] Support for the first and third authors was provided in part by a 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 Advanced Research Projects Agency under Contract N00014-91-J-4052, ARPA, Order 8225. Support for the second author was provided in part by NSF Research Grant CCR-9007851, by Army Research Office Grant DAAL03-91-G-0035, and by the Office of Naval Research and the Advanced Research Projects Agency under Contract N00014-91-J-4052 and ARPA Order 8225.

[2] Department of Computer Science, Brown University, Providence, RI 02912-1910, USA. ss@cs.brown.edu and rt@cs.brown.edu.

[3] Department of Computer Science, Duke University, Durham, NC 27708-0129, USA. jsv@cs.duke.edu.

Received June 26, 1992; revised January 25, 1994. Communicated by C. K. Wong.

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.

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 [14] 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 $n^3/\log n$ processors (using the naive algorithm for matrix multiplication) [14]. 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 $O((Mn)(3 + \omega)/2 \log^3 n)$ 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 undirected graphs the number of processors for the single-source problem can be reduced to $n^{1.5}/(\log^3 n)$, while keeping the parallel time down to $O(\log^3 n)$, by using the nested dissection technique of Pan and Reif [23], [24]. Further improvements [6], [15] have generalized these ideas to include directed planar graphs while keeping the time and processor bounds the same. In recent work Klein and Subramanian [17] have given a linear-processor polylog-time algorithm for finding single-source shortest paths in planar digraphs. However the algorithm in [17] has a polylogarithmic running time with a large exponent.

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* [28] 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 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 [7].

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). Apostolico *et al.* [3] gave an algorithm to compute shortest paths in a grid in $O(\log^2 n)$ time with $O(n)$ processors. 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 algorithms 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.

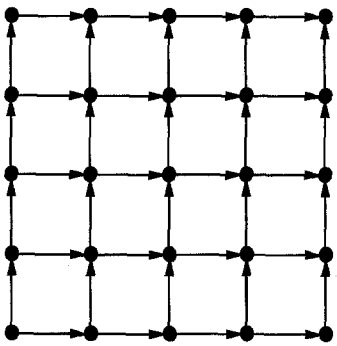

Fig. 1. An example of a grid digraph.

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 p subsets, called *layers*, $l_1, \ldots, l_p$ such that all edges of G are between consecutive layers. Given p parallel lines consecutively numbered in the plane, a *p-line embedding* of G is a drawing such that:

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

A *planar p-line embedding* is a p-line embedding without crossings. G is a planar layered digraph if it admits a planar p-line embedding. Figure 2 gives an example

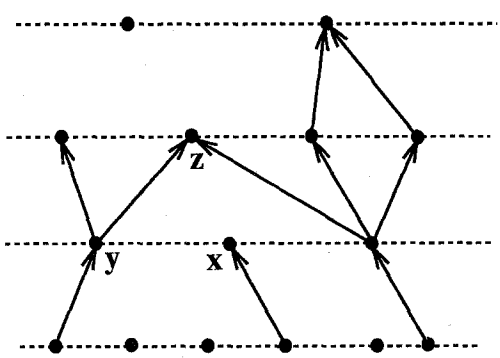

Fig. 2. A planar layered digraph.

of a layered graph with a planar five-line embedding. Layered graphs have been studied under the name of *proper hierarchies* by Wilson [29]. Di Battista and Nardelli [9] give efficient algorithms to test if a layered digraph with only one source is planar. Recently Kosaraju [8] has developed an efficient parallel algorithm to evaluate planar layered circuits.

We can transform any planar layered digraph G into a planar layered digraph with exactly one source and one sink in the following manner: We first introduce two new vertices s and t and put them in two new layers, one at the beginning and one at the end. Then, for each sink x in layer i, we find the closest node y to its left that is attached to a node in layer $i + 1$. We then let z be the rightmost node in layer $i + 1$ that is attached to y (see Figure 2 for an example). If there is no such node y, then we set z to be the leftmost node in layer $i + 1$. To make x a nonsink vertex we introduce an infinite weight edge from x to z. For example, Figure 3 shows how to transform the planar layered digraph in Figure 2 into a single-source single-sink planar layered digraph. The dotted edges are the infinite weight edges. It is not hard to see that these infinite weight edges do not alter any shortest paths or destroy planarity. This computation can be accomplished in $O(\log n)$ time with $n/\log n$ processors using standard techniques (see, for example, [16]). 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

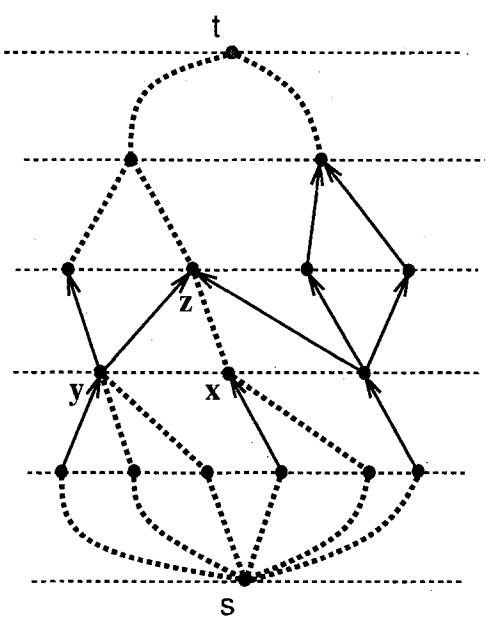

Fig. 3. Adding extra edges to create a planar layered st-graph.

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.

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 the techniques of Tamassia and Vitter [27]. □

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.* [21].

We now define the concept of a *left* ordering of the vertices in a planar *st*-graph, which proves useful in our algorithm. This ordering was introduced by Tamassia and Preparata [26]. 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 [27] give optimal EREW algorithms to construct the left ordering of a planar *st*-graph.

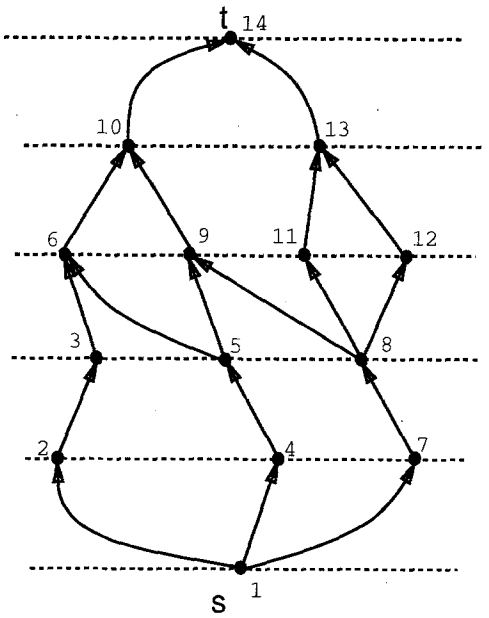

Fig. 4. Left ordering of a planar *st*-graph.

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 $|A_i| \delta n$.

Lipton and Tarjan [22] proved that any planar graph with n vertices has a $\sqrt{8n}$-separator that $\frac{2}{3}$-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 which uses n processors if the breadth-first search tree of the graph is already known was given by Miller [18]; an improved version of the algorithm which uses randomization to find a cycle separator with $n^{1+\varepsilon}$ processors was given by Gazit and Miller [13]. Randomized parallel algorithms to find small separators for more general undirected graphs were given by Miller and Thurston [19]. 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 underyling undirected graph does not guarantee an efficient resursive solution. We get into trouble because even though we can use the separator to divide the original problem into two roughly 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 endpoints 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. A division of G into one or more pieces is a *one-way division* if the separator X used to divide G is a one-way separator.

Grid digraphs, for instance, have one-way separators of size $\sqrt{n}$ that $\frac{1}{2}$ split the graph. In fact the shortest-path algorithm by Apostolico *et al.* [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 crux of the algorithm lies in finding one-way separators. We show how to find one-way separators X_1, X_2, and X_3 such that using three one-way divisions we can partition G into at most four pieces A, B, C, and D, none of which have more than two-thirds of the vertices in G. We then show how this division can be used to formulate a recursive solution to the single-source shortest-path problem. To show the existence of such separators we need the following lemma, which follows directly from the arguments of Lipton and Tarjan [22].

LEMMA 1. *Let G be any n-vertex planar layered st-graph containing layers 1 through p. 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 $p + 1$ be an additional layer containing no vertices. Given any layer j, a layer $i \leq j$ exists 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 a layer $k \geq j$ exists such that $n_k + 2(k - j - 1) \leq 2\sqrt{n - t_j}$. In particular we let i (resp. k) be the layer which minimizes the function $n_i + 2(j - i) \leq 2\sqrt{t_j}$ (resp. $n_k + 2(k - j - 1) \leq 2\sqrt{n - t_j}$).*

THEOREM 2. *Given an n-vertex planar layered st-graph G containing layers 1 through p, at most three one-way separators X_1, X_2, and X_3 exist such that $X = X_1 \cup X_2 \cup X_3$ $\frac{2}{3}$-splits G into up to four pieces. Furthermore, $|X| = O(\sqrt{n})$.*

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 k be layers as in Lemma 1. We let X_1 be the set of vertices S_i in layer i, and let X_2 be the set of vertices S_k in layer k.

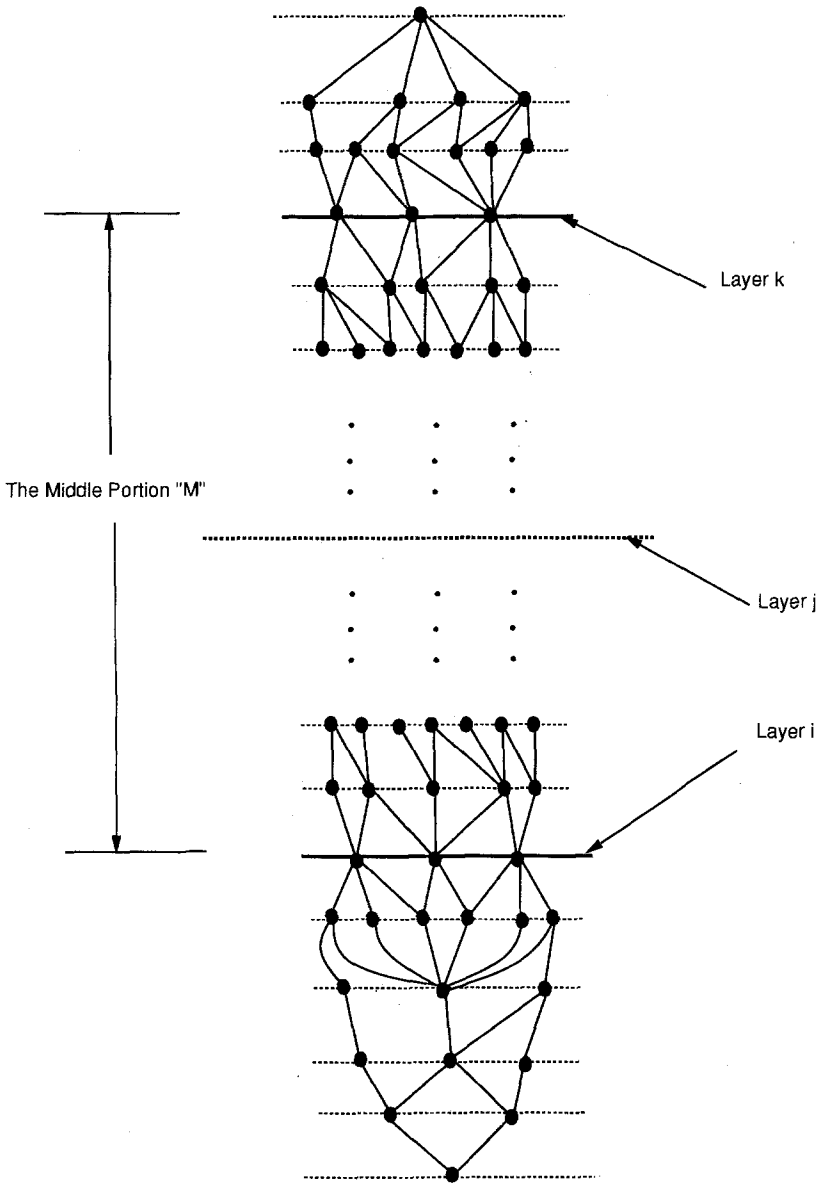

Fig. 5. Slicing the graph into three parts.

Let A be the part of G above k, let B be 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 k to t' and from s' to all the vertices in layer i (see

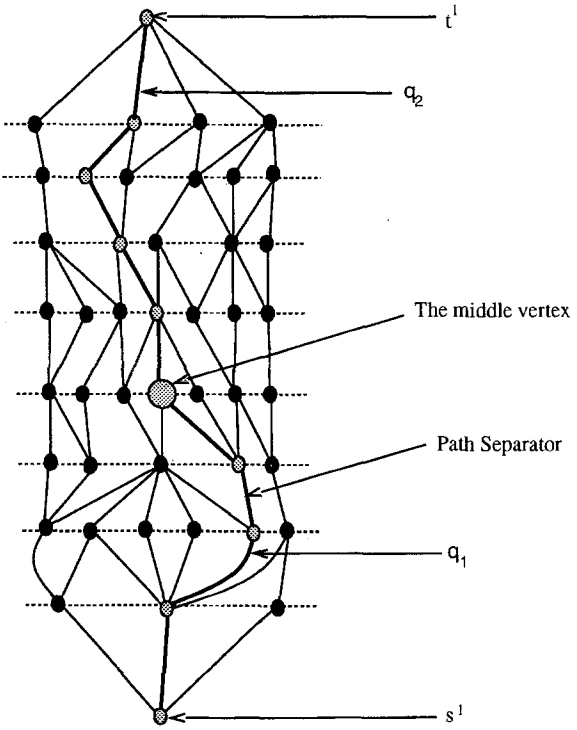

Fig. 6. Finding a path separator in the middle graph.

Figure 6). Let m be the middle vertex in the left ordering L_1 or G', and let q 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 q splits G' into two equal parts C and D. We let X_3 be the set of vertices on q.

We now claim that X_1, X_2, and X_3 are one-way separators and that $X = X_1 \cup X_2 \cup X_3$ is an $O(\sqrt{n})$ separator that $\frac{2}{3}$-splits G. It is easy to see that none of the pieces have more than $2n/3$ vertices; therefore X $\frac{2}{3}$-splits G. By construction $|q| \leq k - i + 1$, therefore, $|X| \leq |X_1| + |X_2| + |X_3 = n_i + n_k + |n_k + |q| \leq n_i + n_k + (k - i + 1) \leq 2(\sqrt{t_j} + \sqrt{n - t_j}) \leq 2(\sqrt{n/2} + \sqrt{n/2}) = 2\sqrt{2n}$.

To prove that X_1, X_2, and X_3 form one-way separators we note the following: By the definition of planar layered digraphs no path in G intersects with vertices in S_i or S_k more than once (see Figure 5); therefore X_1 and X_2 are one-way separators. To look at the interactions of paths in G with the path q, we look at the two subpaths q_1 and q_2 of q, where q_1 is the portion below the middle vertex and q_2 is the portion above (see Figure 6). Let C be the subgraph to the left of q and let D the subgraph to the right of q. By construction that we can see that no path can cross from D to C since no path can enter q_1 from the right, and no path can leave q_2 from the left. Therefore, X_3 is also 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.

1. For very 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 k.
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 "leftmost outgoing and rightmost 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 separator X of Theorem 2 and can be implemented in time* $O(\log n)$ *using* $n/\log n$ *processors on an EREW PRAM.*

PROOF. Steps 1–4 can be done using standard techniques (see [16]). Steps 5 and 6 can be completed in $O(\log n)$ time using the techniques of Tamassia and Vitter [27]. □

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 from s to t. We construct planar layered *st*-graphs from the pieces A, B, C, and D by introducing new source and sink vertices and adding edges from the top and 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 of Lipton and Tarjan [22] 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* $0 \leq \varepsilon \leq 1$. *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* $O(\sqrt{n/\varepsilon})$.

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 pices, each of size at most $2k/3$, where P has k vertices. However, we want to view the splits in terms of one-way separators X_1, X_2, and X_3, respectively. Each such division partitions the graph into two pieces, and the four pieces we obtain after the three divisions have no

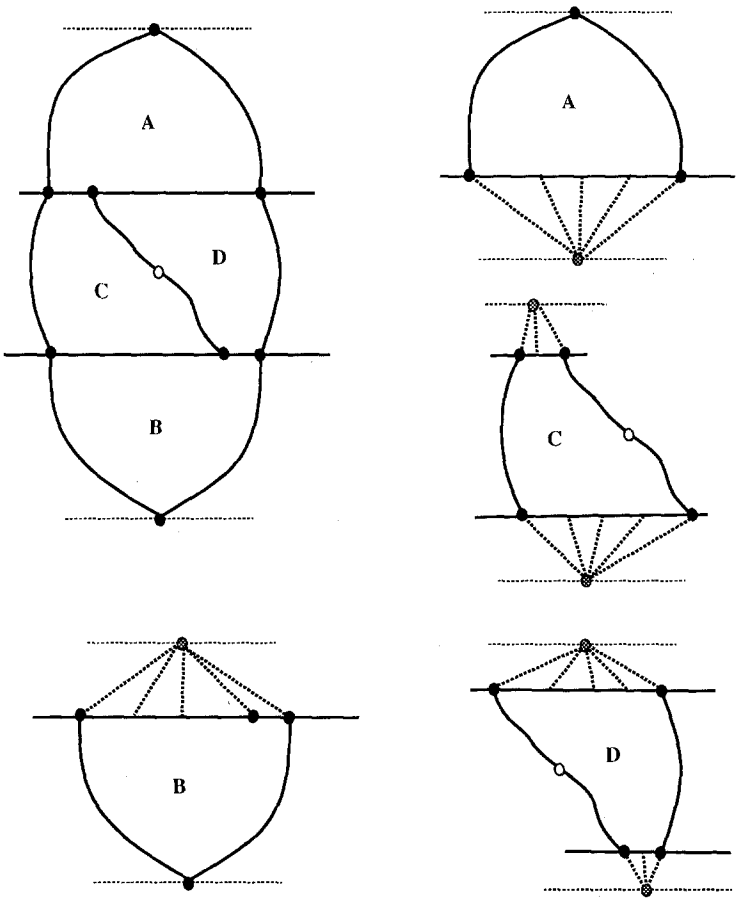

Fig. 7. A schematic view of the four pieces obtained by using the separator.

more than $2k/3$ vertices each. 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 one-way division. The ith level in the tree consists of all the pieces that are at a distance i from the root. It is easy to see that the total number of levels in the RDT is $O(\log n)$ since once every three levels the number of vertices in any piece goes down by a factor of $\frac{2}{3}$. We now claim that at each 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 $\varepsilon = (\frac{2}{3})^{(i-1)/3}$; 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 $O(\sqrt{n/\varepsilon})$. 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 $2\sqrt{2\varepsilon n}$, 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, the total number of source–sink pairs introduced is at most $2\sqrt{2\varepsilon n}\,O(\sqrt{n/\varepsilon}) = O(n)$.

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 two 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.

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 the RDT. Let n_p denote the number of vertices in P. 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 Z. Let n_s, n_t, and n_z be the number vertices in S, T, and Z, respectively.

We know from Theorem 2 that Z is a one-way separator that is either a layer of vertices or a speical "rightmost-edge-in, leftmost-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 Z, and from vertices in Z 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 Z whether it is in the shortest path from a to b or not. This can be done in constant time with $|S|$ processors in a CRCW PRAM. However, a little thought reveals that such a naive appproach would require too many processors. Assume without loss of generality that $n_s \leq n_t$. We first 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 $\log^2 n$ time with $n_s n_t + n_s n_z$ processors. We make use of the following definition and lemma.

DEFINITION 6. Let a and b be two vertices separated by a one-way separator $Z \subset V$ 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 let b_2 be vertices in T. 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 $l(a, b_1)$ in the left ordering.*

PROOF. We give a proof by contradiction for the case when Z is a path; the proof when Z is a layer is similar. Let $m = l(a, b_2)$ and $m_1 = l(a, b_1)$. Let us suppose m occurs before m_1 in the left ordering of P (see Figure 8). By 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 sources vertices to a single sink vertex. We now show how to use Lemma 4 to do the patching up across boundary Z with $n_s n_t + n_s n_z$ processors. For every vertex $a \in S$ we use $n_t + n_z$ 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 $P(a, Z, T)$ where we are given the shortest paths from a to every vertex in Z and between all pairs of vertices (z, b), where z is in Z and b is in T. Our goal

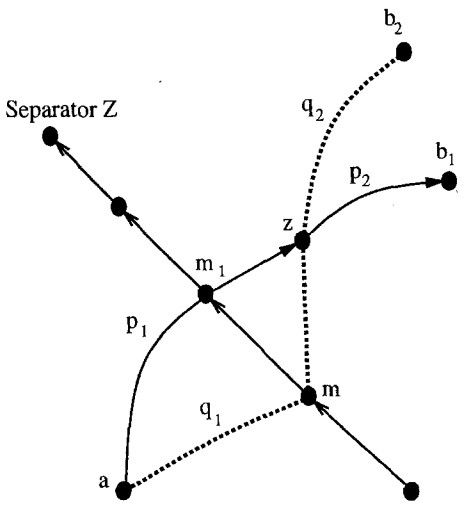

Fig. 8. Shortest paths from a are noncrossing.

is to determine the shortest paths from a to every vertex $b \in T$. The algorithm to solve $P(a, Z, T)$ is as follows:

1. Let b_m be the middle vertex in the left ordering of T.
2. Determine the vertex z_m such that $z_m = l(a, b_m)$.
3. If $n_z = 1$ 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_{12} such that $T_1 = \{b \in T | b < b_m\}$ and $T_2 = T - T_1$. Similarly divide Z into two corresponding sets Z_1 and Z_2 such that $Z_1 = \{z \in Z | z_m\}$ and $Z_2 = Z - Z_1$.
5. Recursively solve the two subproblems $P(a, Z_1, T_1)$ and $P(a, Z_2, T_2)$.
6. For all vertices $b \in T_1$ compare the shortest path from a to b through Z_1, with the one through z_m, and pick the smaller one.

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

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 Z_1, and for all $b \in T_1$ the shortest path from a to b either passes through Z_1 or through z_m. We can therefore recursively divide the problem into two subproblems $P(a, Z_1, T_1)$ and $P(a, Z_2, T_2)$, 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 $n/\log n$ processors [27]. Thus, we can determine the orderings of S and T in $O(\log n_p + \log n)$ time with $\sqrt{n_p}/\log n$ processors. The time complexity of the various steps are as follows:

1. Step 1 can be performed in $O(\log n_t + \log n)$ time with $n_t/\log n$ processors in a EREW PRAM.
2. Step 2 can be performed in time $O(\log n_z + \log n)$ with $n_z/\log n$ 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 $O(\log n_t + \log n_z + \log n)$ time with $(n_t + n_z)/\log n$ 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 $O(\log n_p + \log^2 n_t + \log n_z + \log n)) = O(\log^2 n)$.
6. Step 6 can be done in $O(\log n)$ time with $n_t/\log n$ processors in a CREW PRAM.

To see the processor complexity of the recursive subproblems we note that the two subproblems are solved in parallel partitioning the processors between them. Subproblem $P(a, Z_1, T_1)$ gets $(|Z_1| + |T_1|)/\log n$ processors, while subproblem

$P(a, Z_2, T_2)$ gets $(|Z_2| + |T_1|)/\log n$ processors. The total number of processors needed is $n_s(n_z + n_t)/\log n + n_p/\log n$.

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

PROOF. The proof immediately follows from Lemma 2 and Theorems 3 and 4. □

4. The Tube Minima Problem and a More Efficient Solution. 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 $k \in Z$. 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, $\Theta_{A'}$ for every submatrix A' of A should also satisfy 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 Aggarwal and Park as the *tube minima* problem and has many applications in computational geometry [1]. They give the following bounds for calculating the tube minima of an $\sqrt{n} \times \sqrt{n} \times \sqrt{n}$ *array:*

THEOREM 6. *The tube minima of an* $\sqrt{n} \times \sqrt{n} \times \sqrt{n}$ *array can be found in time* $O(\log n)$ *and work* $O(n)$ *in a CREW PRAM.*

Similarly for the CRCW model of computation Atallah [4] has given an algorithm with the following bounds:

THEOREM 7. *The tube minima of an* $\sqrt{n} \times \sqrt{n} \times \sqrt{n}$ *array can be found in time* $O(\log \log n)$ *and work* $O(n)$ *in a CRCW PRAM.*

We can use these algorithms in the patch-up stage to get improved algorithms. To do that we need to make a slight change to our recursive decomposition. In our current decomposition tree we only worry about the total number of source–sink pairs at any level, and not about the number of boundary vertices in any one particular region. Therefore, our division does not guarantee that the

number of recursive subproblems in each piece is small. It is possible to do the division so that the number of boundary vertices in any piece P (at any level) is $O(\sqrt{n_p})$, where the number of vertices in P is n_p. This can be accomplished by using the following idea due to Frederickson [11]: Mark the boundary vertices and use the separator algorithm from Theorem 3 once, to divide the marked vertices (into small pieces), at every stage. Our separator algorithm can be easily modified to divide a collection of marked vertices. This change in our division strategy guarantees that at any stage each piece has a small number of boundary vertices. In particular, we have the following lemma, which follows from the arguments in [11].

LEMMA 5. *Let G be an n-vertex planar layered st-graph, and let $0 \leq \varepsilon \leq 1$. Using the separator X defined in Theorem 2 we can divide the graph into $\Theta(n/r)$ regions, such that each regions has at most r vertices, and there are only $O(n/\sqrt{r})$ boundary vertices in all. Furthermore, none of the regions contain more than $O(\sqrt{r})$ boundary vertices.*

Using this division strategy the patch-up problem for any parent piece P at any stage can be solved as a tube-minima problem on an $O(\sqrt{n_p} \times \sqrt{n_p} \times \sqrt{n_p})$ array. We therefore get the following bounds for the shortest-path problem:

THEOREM 8. *Given an n-vertex planar layered st-graph G with nonnegative edge weights, we can determine the shortest path from s to t in time $O(\log n)$ with $n/\log n$ processors in a CREW PRAM, and in time $O(\log n \log\log n)$ with $n/\log\log n$ processors in a CRCW PRAM.*

5. Conclusions. We have given a linear processor CREW algorithm for determining the shortest paths in a planary 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 similar approach would give a linear processor solution. As evidence we present the graph in Figure 9 that cannot be decomposed into smaller pieces (that are a constant fraction smaller in weight) by using one-way separators of size $O(\sqrt{n})$. Klein and Subramanian [17] have to some extent resolved this problem by presenting a linear-processor polylog-time algorithm for finding single-source shortest-path problems in planar digraphs. However, their algorithm has a polylogarithmic running time with a large exponent. It may be possible to use a combination of the ideas presented in this paper along with the ones in [17] to get a linear-processor algorithm that has a better running time.

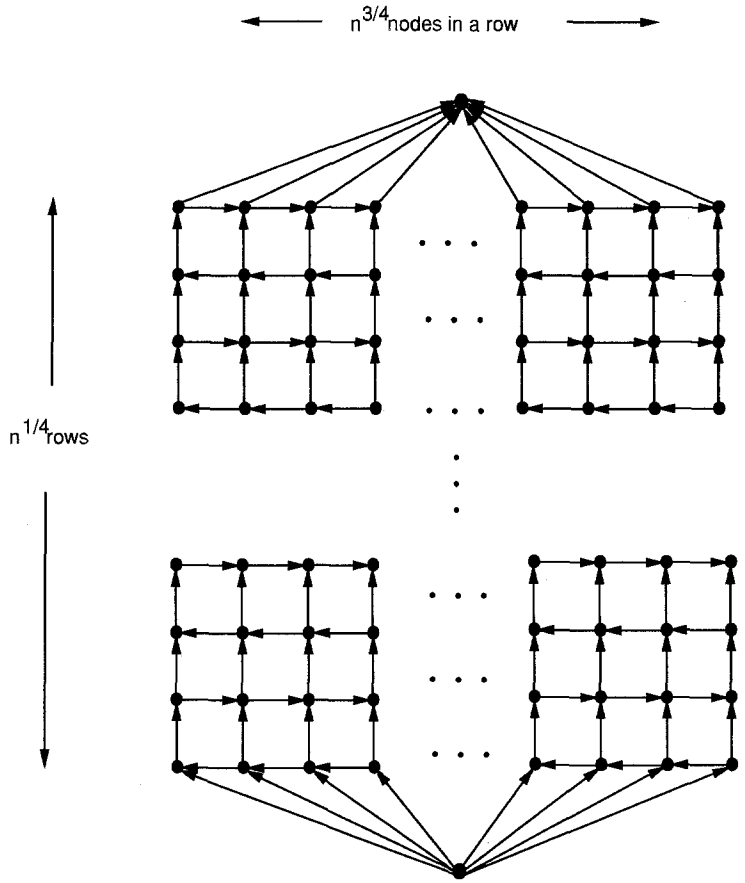

Fig. 9. A bad planar *st*-graph.

Acknowledgments. We would like to thank Alok Aggarwal and Mike Atallah for showing us that we can get better bounds by using the algorithms for the tube minima problem, and for bringing to our attention Frederickson's technique for maintaining a small boundary [11].

References

[1] A. Aggarwal and J. Park, Notes on searching multidimensional monotone arrays, *Proc. 29th Annual IEEE Symposium on Foundations of Computer Science*, 1988, pp. 496–512.

[2] N. Alon and Z. Galil, On the exponent of the all pairs shortest path problem, *Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science*, 1991, pp. 569–575.

[3] A. Apostolico, M. J. Atallah, L. Larmore, and H. S. Mcfaddin, Efficient parallel algorithms for string editing and related problems, *SIAM Journal on 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, pp. 193–200.

[5] R. Bellman, On a routing problem, *Quarterly Journal of Applied Mathematics*, **16** (1958), 87–90.
[6] E. Cohen, Efficient parallel shortest-paths in digraphs with a separator decomposition, *Proc. 5th Annual Symposium on Parallel Algorithms and Architectures*, 1993, pp. 57–67.
[7] R. Cole and U. Vishkin, Optimal parallel algorithms for expression tree evaluation and list ranking, *Proc. Third Aegean Workshop on Computing*, 1988, pp. 91–100.
[8] A. L. Delcher and S. R. Kosaraju, An NC algorithm for evaluating monotone planar circuits, Manuscript.
[9] G. Di Battista and E. Nardelli, An algorithm for testing planarity of hierarchical graphs, *Proc. Workshop WG86*, Bernierd, June 1986 (1987), pp. 277–289.
[10] E. W. Dijkstra, A note on two problems in connexion with graphs, *Numerische Mathematik*, **1** (1959), 269–271.
[11] G. N. Frederickson, Fast algorithms for shortest paths in planar graphs, with applications, *SIAM Journal on Computing*, **16** (1987), 1004–1022.
[12] M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *Journal of the Association for Computing Machinery*, **34** (1987), 596–615.
[13] H. Gazit and G. L. Miller, A parallel algorithm for finding a separator in planar graphs, *Proc. 28th Annual IEEE Symposium on Foundations of Computer Science*, 1987, pp. 238–248.
[14] M. Gondran and M. Minoux, in *Graphs and Algorithms*, Wiley Interscience, New York, 1984.
[15] Y. Han, V. Pan, and J. H. Reif, Efficient parallel algorithms for computing all pair shortest paths in directed graphs, *Proc. Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*, 1992, pp. 353–362.
[16] R. M. Karp and V. Ramachandran, A survey of parallel algorithms for shared memory machines, in *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), North-Holland, Amsterdam, 1990, pp. 871–941.
[17] P. N. Klein and S. Subramanian, A linear-processor polylog-time algorithm for shortest-paths in planar graphs *Proc. 1993 IEEE Symposium on Foundations of Computer Science*, 1993, pp. 259–270
[18] G. L. Miller, Finding small simple cycle separators for 2-connected planar graphs, *Journal of Computer and System Sciences*, **32** (1986), 265–279.
[19] G. L. Miller and W. Thurston, Separators in two and three dimensions, *Proc. 22nd Annual ACM Symposium on Theory of Computing*, 1990, pp. 300–309.
[20] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.
[21] A. Lempel, S. Even, and I. Cederbaum, An algorithm for planitary testing of graphs, *Theory of Graphs, Proc. Internat. Symposium*, 1966, pp. 215–232.
[22] R. J. Lipton and R. E. Tarjan, Applications of a planar separator theorem, *SIAM Journal on Computing*, **9** (1980), 615–627.
[23] V. Pan and J. H. Reif, Fast and efficient solution of path algebra problems, *Journal of Computer and System Sciences*, **38** (1989), 494–510.
[24] V. Pan and J. H. Reif, The parallel computation of minimum cost paths in graphs by stream contraction, *Information Processing Letters*, **40** (1991), 79–83.
[25] G. Shannon and F. Wan, Subdividing Planar Graphs in Parallel, Technical Report, Department of Computer Science, Indiana Universty, 1991.
[26] R. Tamassia and F. P. Preparata, Dynamic maintenance of planar digraphs, with applications, *Algorithmica*, **5** (1990), 509–527.
[27] R. Tamassia and J. S. Vitter, Parallel transitive closure and point location in planar structures, *SIAM Journal on Computing*, **20** (1990), 708–725.
[28] J. Valdes, R. E. Tarjan, and E. L. Lawler, The recognition of series parallel digraphs, *SIAM Journal on Computing*, **11** (1982), 298–313.
[29] S. Whitesides, Forms of hierarchy: a selected bibliography, *General Systems*, **14** (1969), 3–15.