1992

Shortest Path Computations in Source- deplanarized Graphs

Greg N. Frederickson
Purdue University, gnf@cs.purdue.edu

Susanne E. Hambrusch
Purdue University, seh@cs.purdue.edu

Hung-Yi Tu

Report Number:
92-100

SHORTEST PATH COMPUTATIONS IN SOURCE-DEPLANARIZED GRAPHS

Greg N. Frederickson
Susanne E. Hambrusch
Hung-Yi Tu

CSD-TR-92-100
May 18, 1992

Shortest Path Computations in Source-deplanarized Graphs

Greg N. Frederickson *
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Susanne E. Hambrusch †
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Hung-Yi Tu ‡
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

May 18, 1992

Abstract

We consider a class of non-planar graphs that arises in VLSI layout compaction and show that a number of shortest path problems on these graphs can be solved in the same time as the corresponding problem in a planar graph.

Keywords: Analysis of algorithms, shortest paths, planar graphs.

*Research supported in part by ONR under contract N00014-86-K-0689, and by NSF under Grant CCR-90-01241.

†Research supported in part by ONR under contracts N00014-84-K-0502 and N00014-86-K-0689, and by NSF under Grant MIP-87-15652.

‡Research supported in part by NSF under Grant MIP-87-15652 and ONR under contract N00014-84-K-0502.

1 Introduction

Finding shortest paths is a fundamental problem in applied graph theory. Let G be an n-vertex. m-edge graph. For general graphs the single source shortest paths problem can be solved in $O(m + n \log n)$ time by using Dijkstra's algorithm with a Fibonacci heap implementation [3]. A number of special classes of graphs, including planar graphs, are known to have faster algorithms. In this note we consider source-deplanarized graphs, a class of non-planar graphs that arises in VLSI layout compaction and job scheduling [4, 5]. In the compaction application one solves single source problems on a sequence of source-deplanarized graphs where two consecutive graphs differ only in the weights associated with certain edges. We show that a number of shortest path problems on source-deplanarized graphs can be solved in the same time as the corresponding problem in a planar graph. The best bounds for planar graphs are the following [2]. The single source problem can be solved in $O(n\sqrt{\log n})$ time. At a one-time expense of an $O(n \log n)$ preprocessing time, the single source problem can be solved in $O(n)$ time.

A *source-deplanarized graph*, or *s-graph* for short, G_s, is a directed graph obtained by taking a directed, planar graph G with nonnegative edge weights and adding a source s and edges with nonnegative weights from s to vertices in G. Note that this operation can complicate the embedding structure of the graph considerably; the genus of the resulting s-graph can be $\Theta(n)$. We refer to the edges incident on the source s as the *s-edges*. These s-graphs are used in the VLSI layout compaction algorithms of [4] in which single source problems are solved in an on-line fashion on a sequence of s-graphs. The i-th graph in this sequence differs from the $(i + 1)$-st one in the weights associated with s-edges. The shortest path tree for the i-th graph determines the changes in the weights of the s-edges. Solving single source problems faster by allowing a larger preprocessing time is clearly useful in this context.

In Section 2 we show how to determine the shortest path tree rooted at s in $O(n \log^* n)$ time with an $O(n \log n)$ preprocessing time. This algorithm allows us to determine in $O(n \log^* n)$ time the shortest path tree after the weights of an arbitrary number of s-edges have been changed. We also show that the single source shortest paths problem on an s-graph can be solved in $O(n\sqrt{\log n})$ time without preprocessing. During the layout compaction algorithm

described in [4] the weights associated with the s-edges decrease. In Section 3 we show that, if the weight of a single s-edge decreases, the resulting shortest path tree can be determined in $O(n)$ time. This algorithm requires an $O(n \log n)$ preprocessing time.

2 Handling a set of weight changes

In this section we describe our algorithm for solving the single source problem on an s-graph in $O(n \log^* n)$ time with $O(n \log n)$ preprocessing. As already stated, this algorithm will allow us to solve one instance of a single source problem in $O(n \log^* n)$ time after the weights of the s-edges have been modified. Our overall approach is similar to the one used in [2]. We start by describing the concepts needed to understand the algorithm.

As in [2], we assume that a planar graph G has already been transformed so that no vertex has degree greater than 3. A *suitable r-division* divides G into $\Theta(n/r)$ regions, where a region contains two types of vertices, boundary vertices and interior vertices. A region contains at most r vertices and $O(\sqrt{r})$ boundary vertices. Furthermore, each boundary vertex is contained in at most three regions, and any region that is not connected consists of connected components, all of which share boundary vertices with exactly the same set of either one or two connected regions. A suitable r-division can be found in $O(n \log r + \frac{n}{\sqrt{r}} \log n)$ time [2, Lemma 4].

Frederickson's basic algorithm consists of two phases, a *preprocessing phase* and a *search phase*. Let r be a given parameter. For two boundary vertices v and w belonging to the same region R_i, let $d_i(v, w)$ be the length of the shortest path from v to w lying entirely within region R_i. The preprocessing step computes, for each region R_i, the shortest path between every pair of its boundary vertices. The search phase consists of two parts, the *main thrust* and the *mop-up*. For each boundary vertex v, let $\rho(v)$ be the current shortest distance from source s to v, with $\rho(s) = 0$ and $\rho(v) = \infty$ initially. At each iteration of the main thrust, the vertex v with minimum $\rho(v)$ is chosen. For every boundary vertex w in region R_i, $\rho(w)$ is updated; i.e., $\rho(w) = \min\{\rho(w), \rho(v) + d_i(v, w)\}$. At the termination of the main thrust, the length of the shortest path from the source s to each boundary vertex is known. The mop-up determines the shortest path distances from the source s to the interior vertices.

The time complexity is established as follows. Using Dijkstra's algorithm [1] for determining the shortest paths between every pair of the boundary vertices results in $O(n\sqrt{r}\log r)$ time for all regions. Using data structures and techniques described in [2], the main thrust can be accomplished in $O(n + \frac{n}{\sqrt{r}}\log n)$ time. The mop-up takes $O(r\log r)$ time for each of the $\Theta(\frac{n}{r})$ regions, yielding $O(n\log r)$ time altogether. Choosing $r = \frac{\log n}{\log\log n}$ results in an $O(n\sqrt{\log n}\sqrt{\log\log n})$ time algorithm for the single source problem in a planar graph. To obtain Frederickson's $O(n\sqrt{\log n})$ time bound, the described approach is generalized to two levels of subdivisions. Another type of generalization to $\log^* n$ levels yields the $O(n\log^* n)$ time algorithm, and the clever use of a decision tree technique, together with three levels of subdivisions, give the $O(n)$ time algorithm. Both of these algorithms require an $O(n\log n)$ preprocessing time.

We now return to the single source shortest path computation in s-graphs. Let G be the planar graph obtained by removing from G_s vertex s and the incident s-edges. Our overall approach is to find suitable r-divisions for G and to adapt the planar shortest path algorithm so that the existence of the s-edges is taken into account.

Let $\log^{(k)} n = \underbrace{\log\log\ldots\log}_{k} n$, for $1 \leq k \leq \log^* n$. We next describe a k-level algorithm solving the single source problem on G_s in $O(n\log^{(k)} n)$ time with $O(n\log n)$ preprocessing. For $k = 1$, we simply use Dijkstra's algorithm. For $k \geq 2$, the algorithm operates on k levels of subdivisions. Let $r_i = (\log^{(i)} n)^2$, for $1 \leq i \leq k$, and let $R_{i,j}$ be the j-th region in the level i subdivision. We view G as the single region forming the level 0 subdivision i.e., $G = R_{0,1}$. The preprocessing necessary for the k-level algorithm involves only graph G and is as in [2, Section 7]. The preprocessing step determines for every region $R_{i,j}$ of a level i subdivision a suitable r_{i+1}-subdivision, $0 \leq i < k$. For every pair of boundary vertices v and w of region $R_{i,j}$ it determines the length of the shortest path between v and w lying within region $R_{i,j}$.

The search phase of the k-level algorithm consists of an initialization step, a main thrust, and a mop-up step. Let $G_s(i,j)$ be the subgraph of G_s induced by the vertices in region $R_{i,j}$ and vertex s. For any two boundary vertices v and w in region $R_{i,j}$, let $d_{i,j}(v,w)$ be the length of the shortest path from v to w in $G_s(i,j)$. For any boundary vertex u in region $R_{1,j}$, let $\rho_j(u)$ be the length of the shortest path from s to u in subgraph $G_s(1,j)$. We also refer to

the $\rho_j(\cdot)$-entries as the *restricted distances.*

The initialization step computes for every boundary vertex u of region $R_{1,j}$ the restricted distance $\rho_j(u)$. The objective of the main thrust and the mop-up step is to compute the final $\rho(\cdot)$-entries, i.e., the value of the shortest path from s to every vertex in G_s. We initialize the $\rho(\cdot)$-entries for level 1 boundary vertices u with $\rho(u) = \min_{u \in R_{1,j}} \{\rho_j(u)\}$ and with ∞ for every other vertex. We then run the main thrust on the level 1 boundary vertices. After the main thrust the length of the shortest path from s to any level 1 boundary vertex is know. In order to determine the $\rho(\cdot)$-entries for the interior vertices of the level 1 subdivision, the mop-up step proceeds as follows. It modifies the weights of the s-edges of every graph $G_s(1,j)$ so that for every boundary vertex u of region $R_{1,j}$ the weight of the edge (s,u) equals $\rho(u)$. This modification can introduce new edges or decrease existing weights. We then apply to each so modified graph $G_s(1,j)$ the search phase of the $(k-1)$-level algorithm. Observe that all the subdivisions and shortest path entries (i.e., $d_{i,j}(\cdot,\cdot)$-entries) between boundary vertices needed by the search phase of the $(k-1)$-level algorithm have already been determined by the preprocessing step of the k-level algorithm. We are now ready to prove the following result.

Theorem 2.1 *Given an source-deplanarized graph, the shortest path tree rooted at source s can be determined in $O(n \log^* n)$ time with $O(n \log n)$ preprocessing.*

Proof: We show that the k-level algorithm described above solves the problem in $O(n \log^{(k)} n)$ time with $O(n \log n)$ preprocessing. The claimed time bound follows for $k = \log^* n$. The correctness of the k-level algorithm follows from the above discussion. The $O(n \log n)$ time bound for generating the k levels of subdivisions in the preprocessing phase follows from [2, Section 7]. The shortest path information between boundary vertices of region $R_{k,j}$ is obtained by applying Dijkstra's algorithm to each region $R_{k,j}$ of the level k subdivision. This uses $O(n \log n)$ time for $k = 1$, and for $k \geq 2$ $O(\frac{n}{\sqrt{r_k}} r_k \log r_k) = O(n\sqrt{r_k} \log r_k)$ time which is $O(n \log n)$. The $d_{i,j}(\cdot,\cdot)$-entries representing the shortest path distances between boundary vertices of region $R_{i,j}$ for $1 \leq i < k$ are obtained by using the entries computed for the level $(i-1)$ subdivision and applying the main thrust. This uses $O(r_i)$ time per level i region and thus $O(n)$ time for all level i regions. Hence, the total preprocessing time is bounded by $O(n \log n)$.

In order to compute the restricted distances $\rho_j(u)$ for every boundary vertex u of region $R_{1,j}$, we apply the search phase of the $(k-1)$-level algorithm to subgraph $G_s(1,j)$. All the subdivisions and $d_{i,j}(\cdot,\cdot)$-entries needed have already been computed by the preprocessing phase of the k-level algorithm. The search phase of the $(k-1)$-level algorithm costs $O(r_1 \log^{(k-1)} r_1)$ time per region and $O(n \log^{(k)} n)$ time totally for all n/r_1 regions. Applying the main thrust to all level 1 boundary vertices costs $O(n)$ time. In the mop-up step we apply the search phase of the $(k-1)$-level algorithm to every subgraph $G_s(1,j)$ whose edge weights have been modified. This costs $O(n \log^{(k)} n)$ time for all n/r_1 graphs. In total, the search phase uses $O(n \log^{(k)} n)$ time. □

Assume we are solving the shortest path problem on a sequence of s-graphs which differ in the weights associated with the s-edges. It follows that, after $O(n \log n)$ preprocessing time, every shortest path tree can be determined in $O(n \log^* n)$ time. When we only solve one instance of a shortest path problem on an s-graph, we are interested in balancing preprocessing and search time and minimizing the total time. This is done by using the 2-level algorithm with $r_1 = \log n$ and $r_2 = (\log \log n)^2$ which results in a total time of $O(n\sqrt{\log n})$.

3 Handling a single weight decrease

When the single source problem is solved during the VLSI layout compaction algorithm described in [4], the weights of the s-edges decrease. Let G_s and G'_s be two s-graphs that differ only in the weight associated with one s-edge. In this section we describe an algorithm that, given the shortest path tree for G_s, generates the shortest path tree for G'_s in $O(n)$ time. Our algorithm requires a one-time preprocessing cost of $O(n \log n)$.

Let (s,u) be the s-edge having weight $w_{s,u}$ in G_s and weight $w'_{s,u}$ in G'_s with $w_{s,u} > w'_{s,u}$. Let G be again the planar graph obtained from G_s (resp. G'_s) by removing the s-edges. If the all pairs shortest path information for graph G is available, the length of the shortest path from source s to every node v in G'_s can be determined in $O(n)$ time. Hence, it easy to solve the problem in $O(n)$ time using $O(n^2)$ preprocessing time and $O(n^2)$ space. Recall that the all-pair shortest paths problem in a planar graph can be solved in $O(n^2)$ time [2, Theorem 6].

In order to reduce the preprocessing time and the space, assume that the single source

shortest path information for G_s with s being the source is available. In order to generate the single source shortest path information for G'_s we perform a shortest path computation on G with u as the source. A single source computation on a planar graph can be performed in $O(n)$ time with a one-time expense of an $O(n \log n)$ preprocessing time [2, Theorem 5]. This algorithm uses $O(n2^{c(\log\log\log n)^2})$ space, for some constant c.

The details of our algorithm are now as follows. We preprocess G so that any single source computation on it can be performed in $O(n)$ time. This preprocessing of G consists of determining three levels of subdivisions, building a decision tree for each region of the level 3 subdivision, and computing for every pair of boundary vertices of a region the length of the shortest path lying within this region. These computations can be done in $O(n \log n)$ time as described in [2]. The preprocessing step also includes the computation of the ρ-entries associated with G_s. These entries can be computed, for example, in $O(n\sqrt{\log n})$ time using the 2-level algorithm mentioned in the last paragraph of Section 2.

The objective of the search phase is the computation of the shortest path entries for G'_s. Let $\rho'(v)$ be the length of the shortest path from s to v in graph G'_s. In order to determine the ρ'-entries we perform a single source shortest path computation on G with u being the source. Let $\rho_u(v)$ be the length of the shortest path from u to v in G. The ρ'-entries can now be computed as follows:

$$\rho'(v) = \min\{\rho(v), w'_{s,u} + \rho_u(v)\}.$$

Theorem 3.1 *Given a source-deplanarized graph, the change in the shortest paths distances caused by a sequence of on-line changes with each change decreasing the weight of one s-edge, can be determined in $O(n)$ time per change with an $O(n \log n)$ preprocessing time.*

Proof: We use the method described above. The time bound is established as follows. Generating the shortest path tree for G with u as the source is done in $O(n)$ time [2, Theorem 6]. The value of the ρ'-entries is then generated in another $O(n)$ steps. Thus, the search phase has an overall $O(n)$ time bound. □

References

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] G.N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Computing*, 16(6):1004–1022, December 1987.

[3] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of ACM*, 34(3):596–615, 1987.

[4] S. E. Hambrusch and H.-Y. Tu. A framework for 1-d compaction with forbidden region avoidance. *Computational Geometry: Theory and Applications*, 1(4):203–226, April 1992.

[5] C.E. Leiserson and J.B. Saxe. A mixed-integer linear programming problem which is efficiently solvable. *Journal of Algorithms*, 9(1):114–128, March 1988.