1988

# Planar Graph Decomposition and All Pairs Shortest Paths

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

Report Number:
88-788

# PLANAR GRAPH DECOMPOSITION AND ALL PAIRS SHORTEST PATHS

Greg N. Frederickson

CSD-TR-788
July 1988
(Revised October 1989)

# PLANAR GRAPH DECOMPOSITION AND ALL PAIRS SHORTEST PATHS*

(revised October 1989)

Greg N. Frederickson

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

* This research was supported in part by the National Science Foundation under Grants DCR-8320124 and CCR-8620271, and by the Office of Naval Research under contract N00014-86-K-0689.

**Abstract.** An algorithm is presented for generating a succinct encoding of all pairs shortest path information in a directed planar graph $G$ with real-valued edge costs but no negative cycles. The algorithm runs in $O(pn)$ time, where $n$ is the number of vertices in $G$, and $p$ is the minimum cardinality of a subset of the faces that cover all vertices, taken over all planar embeddings of $G$. The algorithm is based on a decomposition of the graph into $O(pn)$ outerplanar subgraphs satisfying certain separator properties. Linear-time algorithms are presented for various subproblems including that of finding an appropriate embedding of $G$ and a corresponding face-on-vertex covering of cardinality $O(p)$, and of generating all pairs shortest path information in a directed outerplanar graph.

**Key words and phrases.** all pairs shortest paths, approximation algorithm, compact routing table, graph embedding, NP-completeness, outerplanar graph, planar graph, succinct encoding.

A fundamental problem in graph algorithms is that of determining shortest path information in a graph [AHU, DP]. Efficient algorithms for various versions of this problem have been proposed [D, Fl, Fs2, Fm, FT, W], with recent emphasis on exploiting topological features of the input graph, such as edge sparsity and planarity [FT, Fs2]. Consider the all pairs shortest paths problem on a directed graph with real-valued edge weights, but no negative cycles. In this paper we introduce a new approach for this problem, using a succinct encoding of shortest path information based on the topology of the graph. We present algorithms that handle $n$-vertex planar graphs in time that ranges from $O(n)$ up to $O(n^2)$ as the topological properties of the graphs become more complex. By encoding shortest path information in what we call compact routing tables, we avoid a lower bound of $\Omega(n^2)$ time that would be needed if the output were required to be in the form of $n$ shortest path trees.

Let $\hat{G}$ be a plane embedding of a planar graph $G$. We call a set of faces of $\hat{G}$ that together cover all the vertices a *face-on-vertex covering*. An example of an embedded planar graph $\hat{G}$ and a face-on-vertex covering of $\hat{G}$ is shown in Figure 1. Let $p$ be the minimum cardinality of any face-on-vertex covering over all plane embeddings of $G$. The value of $p$ ranges from 1 up to $\Theta(n)$, depending on the planar graph. Given planar graph $G$, but no embedding, our algorithm constructs compact routing tables for all pairs shortest paths in a directed planar graph in $O(pn)$ time. Previous algorithms had performance of $O(n^3(\log\log n)^{1/3}/(\log n)^{1/3})$ for general graphs [Fm], $O(nm + n^2\log n)$ for sparse graphs [FT], $O(n^2)$ for planar graphs [Fs2], and $O(n)$ for undirected outerplanar graphs [FJ1], where $m$ is the number of edges in the graph.

Our choice of output in the form of compact routing tables is natural, as shortest path information in this form is useful in space-efficient methods for message routing in distributed networks [FJ1, FJ2]. In the conclusion we shall discuss an alternative encoding that costs only $O(n+p^2)$ time to generate. We also give an algorithm that identifies all edges that violate the generalized triangle inequality in $O(n+p^2)$ time.

We identify several nice structural properties of planar graphs. Given a face-on-vertex covering of cardinality $p'$, we identify a decomposition of a planar graph into $O(p')$ particularly appropriate subgraphs, called hammocks. We prove a *monotonicity property*, which characterizes the difference of the distance from a vertex to two other vertices as the vertex moves around a face in the embedding. To handle our all pairs shortest paths problem, we identify and solve the following four subproblems (the latter two of which are approximation problems for NP-hard problems!):

1. Suppose we are given a directed outerplanar graph with real-valued edge costs, but no negative cycles. We present an algorithm that determines shortest path information in $O(n)$ time. The approach in [FJ1] for undirected outerplanar graphs does not work in the directed case, since for directed outerplanar graphs there is no property comparable to the reflection property.

2. Suppose we are given a directed planar graph $G$ with real-valued edge costs but no negative cycles, Suppose we are also given an embedding $\hat{G}$ of $G$, and a face-on-vertex covering of $\hat{G}$ of cardinality $p'$. We present an algorithm that determines shortest path information in $O(p'n)$ time.

3. Suppose that an embedding $\hat{G}$ of an undirected planar graph $G$ is given, but no

face-on-vertex covering is provided. It has been shown in [FL, BM] that it is NP-complete to determine if there is a face-on-vertex covering of cardinality at most $p'$. We present an algorithm that determines a face-on-vertex covering of cardinality at most twice that of a minimum cardinality face-on-vertex covering for $\hat{G}$. This algorithm is based on an approach in [B], and runs in $O(n)$ time.

4. Suppose we are given an undirected planar graph $G$, but no embedding $\hat{G}$. Note that there are planar graphs for which one embedding has a face-on-vertex covering of cardinality 2, while another embedding has a face-on-vertex covering of minimum cardinality $\Theta(n)$. An algorithm to determine a minimum cardinality face-on-vertex covering and associated embedding is given in [BM], but takes $O(2^p n)$ time. This is too much for our shortest paths application except when $p$ is $\Theta(1)$. We give an algorithm that finds an embedding $\hat{G}$ and a face-on-vertex covering in $\hat{G}$ of cardinality within a constant factor of the minimum cardinality covering for any embedding of $G$. The algorithm uses a decomposition of $G$ into triconnected components [HT1], and runs in $O(n)$ time.

Our algorithm for problem 1 is used in the solution of problem 2, and our algorithm for problem 3 is used in the solution of problem 4. The all pairs shortest paths problem in planar graphs can then be solved as follows. Given a directed planar graph $G$, we first find a good embedding and a good face-on-vertex covering by converting the directed edges to undirected edges, and then applying the algorithm for problem 4. Given the good embedding and the good face-on-vertex covering, we then use the algorithm for problem 2.

Our paper is organized as follows. In section 2 we discuss compact routing tables, and then describe a decomposition of a planar graph. In section 3 we present an algorithm for finding all pairs shortest paths in directed outerplanar graphs. In section 4 we sketch our basic approach for solving all pairs shortest paths in planar graphs, and present the monotonicity property and its application. In section 5 we show how to generate shortest path information between the subgraphs in our decomposition. In section 6 we show how to generate shortest path information within each subgraph in our decomposition. In sections 7 and 8 we describe our approximation algorithm for finding a good embedding and a good face-on-vertex covering. In section 9 we discuss verifying the triangle inequality, and give another encoding of all pairs shortest paths.

A preliminary version of this paper appeared in [Fs3].

## 2. Structure of planar graphs

In this section we first review the notion of compact routing tables. We then define, relative to a given face-on-vertex covering of a planar graph, subgraphs that we term hammocks. Hammocks have several nice properties that make them especially appropriate for use in shortest paths algorithms. A hammock is outerplanar, each hammock shares at most four vertices with the rest of the graph, and the vertices in a hammock form two chains of consecutive vertices along faces in the face-on-vertex covering. Our definition of hammocks leads to a linear-time algorithm for decomposing a planar graph into $O(p')$ hammocks, if the given face-on-vertex covering has $p'$ faces.

We first discuss the idea of compact routing tables, which appears in [FJ1] and is

based on ideas in [SK, vLT]. Let the vertices be assigned names from 1 to $n$ in an appropriate manner to be discussed. For every edge $<v, w>$ incident from any given vertex $v$, let $S(v, w)$ be the set of vertices such that there is a shortest path from $v$ to each vertex in $S(v, w)$ with the first edge on this path being $<v, w>$. A *tie* occurs if there is a vertex $u$ such that there is a shortest path from $v$ to $u$ with the first edge on this path being $<v, w>$ and also a shortest path from $v$ to $u$ with the first edge on this path being $<v, w'>$ for some $w' \neq w$. In the event of ties, an appropriate tie-breaking rule is employed so that for each pair of vertices $v$ and $u \neq v$, $u$ is in just one set $S(v, w)$ for some $w$. Let each set $S(v, w)$ be described as a union of a minimum number of subintervals of $[1, n]$. Here we allow a subinterval to wrap around from $n$ back to 1, i.e., a set $\{i, i+1, \cdots, n, 1, 2, \cdots, j\}$, where $i > j+1$ will be described by $[i, j]$. We call the set $S(v, w)$ described in the form of a minimum number of subintervals of $[1, n]$ the *label* of edge $<v, w>$.

For example, consider an outerplanar graph. (A graph is outerplanar if it can be embedded in the plane such that all vertices are on one face [H].) It was shown in [FJ1] that if the vertices of an undirected outerplanar graph are named in clockwise order around this one face, then each set $S(v, w)$ is a single interval $[l, h]$. Clearly this property also holds for directed outerplanar graphs. A compact routing table for $v$ consists of a list of initial values $l$ of each interval, along with pointers to the corresponding edges. The list is a rotated list [MS, Fs1], and can be searched using a modified binary search.

A linear-time algorithm has been presented in [FJ1] for determining the labels of all edges of an undirected outerplanar graph. (For an undirected graph, each edge has

two labels, one corresponding to each endpoint of the edge, since the edge can be traversed in either direction.) In the next section we give a linear-time algorithm for determining the labels of all edges of a directed outerplanar graph. If the graph is not outerplanar, i.e., $p > 1$, then an edge label $S(v, w)$ can consist of more than one subinterval. A compact routing table will then have an entry for each of the subintervals contained in an edge label at $v$. It can be shown that the total size of all compact routing tables for directed planar graphs is $O(pn)$. (The proof is essentially the same as the proof in [FJ1] for undirected planar graphs.)

For the remainder of the section we discuss a decomposition of an embedded directed planar graph with no self-loops into subgraphs, each of which is outerplanar, and each of which shares at most four vertices with all other subgraphs in the decomposition. Let $\hat{G} = (V, E, F)$ be an embedding of $G$ with a face-on-vertex covering $F'$ of $p'$ faces, where $p' > 1$. To generate the decomposition, we shall first convert the embedded directed graph $\hat{G}$ into an embedded undirected planar graph $\hat{G}_1 = (V_1, E_1, F_1)$ with certain nice properties, along with a face-on-vertex covering $F_1'$ of size $p'$. We shall then identify certain subgraphs in $\hat{G}_1$, and convert these back into the desired subgraphs of $G$. The conversion will be such that $\hat{G}_1$ has no parallel edges, that all faces in $F_1 - F_1'$ are bounded by three edges, that no pair of faces in $F_1'$ share a vertex, and that the boundary of each face is a simple cycle. We call an embedded undirected planar graph that satisfies the above assumptions *neatly prepared.* We describe the structure of $\hat{G}_1$ with respect to $F_1'$, and give a decomposition of $\hat{G}_1$ into $O(p')$ outerplanar graphs. At the end of the section we discuss how to perform the conversions.

In the case that $p' = 2$, there is a special procedure which we discuss subsequently. Otherwise, for $p' > 2$, let the faces in $F_1'$ be indexed with the integers from 1 to $p'$. We label the faces of $F_1 - F_1'$ with a 4-tuple $(i, j, k, r)$. The values $i$, $j$ and $k$ are the indices of the faces in $F_1'$ containing the three vertices of the face. These are ordered so that $k = j$ implies $i = j$. The value $r$ is the number of edges of the face that are shared with faces in $F_1'$. Thus $(i, i, j, 1)$ represents a face that has two vertices on face $f_i$ and one on face $f_j$, with the two vertices on face $f_i$ adjacent via an edge on face $f_i$. Also, $(i, i, i, 2)$ represents a face with three vertices on face $f_i$, with one pair of vertices adjacent via an edge on $f_i$, and a second pair adjacent via a second edge that is also on $f_i$.

We now show how to group the faces together to form hammocks, using two operations: absorption and sequencing. We first perform *absorption*. Consider a pair of faces in $F_1 - F_1'$ that share an edge. Suppose the labels are $(i, i, i, 2)$ and $(i, i, j, r)$, where either $j = i$ and $r = 1$, or $j \neq i$ and $r = 0$. Absorb the first face into the second, and relabel it as $(i, i, j, r+1)$. This is equivalent to performing the following operations on the embedded graph. First contract an edge that the first face shares with face $f_i$. The first face becomes a face bounded by two parallel edges, one of which is shared with the second face. Then delete this edge, effectively merging the faces. Repeat the absorption operation until it can no longer be applied.

Once the absorption operation can be applied no longer, we group the remaining faces by *sequencing*. Identify maximal sequences of faces such that each consecutive pair of faces in the sequence share an edge in common, and all faces have the label $(i, i, j, 1)$ or $(j, j, i, 1)$, for some pair of indices $i$ and $j$. A special case arises if such a sequence of faces extends all the way around one of the faces in $F_1'$, say $f_i$. In this case

there is a vertex on $f_i$ that is contained in both the first and last faces of the sequence. Split this vertex into two vertices. Each such sequence of faces then comprises an outerplanar graph. Expanding the faces that were absorbed into faces in the sequence yields a graph that is still outerplanar. Each such resulting graph is called a *(major) hammock*, so called because it stretches between two faces. The first and last vertices on each of these two faces of the hammock are called the *vertices of attachment*. Any edge that is not included in a major hammock is taken individually to induce a *(minor) hammock*. The set of all major and minor hammocks comprises a *hammock decomposition* of the embedded graph $\hat{G}_1$.

In Figure 2 is an undirected embedded planar graph that was generated from the directed embedded planar graph given in Figure 1. (We discuss this generation later.) Let the faces in the face-on-vertex covering be indexed: $f_1$ covers vertices 1-7, $f_2$ covers vertices 8-11, $f_3$ covers vertices 12-14, $f_4$ covers vertices 15-20, and $f_5$ covers vertices 21-23. The face containing vertices 9, 10, and 11 will have label (2, 2, 2, 2), and the face containing vertices 3, 9, and 11 will have label (2, 2, 1, 0). Absorbing the first face into the second by contracting edge (9, 10) and deleting edge (9, 11) will yield a face with vertices 3, 9, and 11 and label (2, 2, 1, 1). Then a maximal sequence of faces between faces $f_1$ and $f_2$ contains faces with vertex sets {3, 9, 11}, {3, 4, 9}, {4, 9, 8}, {4, 8, 11}, and {4, 5, 11}. (The face {3, 9, 11} in this sequence is the result of absorbing face {9, 10, 11} into the original face {3, 9, 11} of Figure 2. As noted earlier, this resulting face has edge (9, 11) on face $f_2$.) Note that this sequence extends all the way around face $f_2$. Thus the vertex 11 should be split into two vertices, say $11'$ and $11''$. The edges in the corresponding major hammock will be (9, 10), (9, $11''$), (10, $11''$), (3, 9), (3, 4),

(4, 9), (8, 9), (4, 8), (8, 11′), (4, 11′), (4, 5), (5, 11′), and (3, 11″). These are listed in an order of six instances of an edge that can be contracted followed by an edge that can be deleted, culminating with a final edge that remains, as discussed in the proof of the upcoming Lemma 2.2. Note that the vertices of attachment of this hammock are 3, 11″, 5, and 11′.

Note that a major hammock can span between two different sequences of vertices on the same face in $F_1'$, as is shown by the hammock that spans vertices {1, 7, 3}. There can also be two different hammocks spanning between the same pair of faces, as shown by the hammock spanning vertices {5, 6, 7, 16, 15, 20} and the hammock spanning vertices {7, 16, 17}. Other major hammocks span the vertex sets {1, 2, 3, 12, 13, 14}, {17, 18, 19, 20, 21, 22, 23}, and {7, 21, 23}. Note that edges (7, 11) and (11, 16) each induce a minor hammock.

If $p' = 2$, then $\hat{G}_1$ can be decomposed as follows. (Note that after absorbing all possible faces, there would be a cycle of faces rather than a sequence of faces.) Identify a face not in $F_1'$ that contains vertices not all on the same face in $F_1'$. Of the vertices on this face, choose two which are on different faces of $F_1'$. Split each of these vertices into two vertices, and reconnect the edges so that the face not in $F_1'$ and the two faces in $F_1'$ are merged. The resulting graph is outerplanar, and we designate it a major hammock. The vertices of attachment are the four vertices resulting from the splitting.

**Lemma 2.1.** The above algorithm generates a decomposition of a neatly prepared embedded undirected planar graph into hammocks.

**Proof.** We claim that there is a one-to-one correspondence between the edges in $\hat{G}_1$ and the edges in the hammocks of the hammock decomposition of $\hat{G}_1$. If $p' = 2$, then this is clearly true. Thus we consider the case in which $p' > 2$. First it is clear that any edge in $\hat{G}_1$ has at least one corresponding edge in the hammocks of the decomposition, since any edge not in a major hammock is inserted into a hammock of its own. Suppose that there were an edge in $\hat{G}_1$ that is in more than one major hammock. This edge cannot be an edge on some face $f_i$ in $F_1'$, since such an edge is in only one face in $F_1 - F_1'$, and each face in $F_1 - F_1'$ is included in at most one sequence of faces. Thus this edge must be shared by two faces in $F_1 - F_1'$, each of which is included in a different hammock. Let one face have label $(i, i, j, 1)$ after all absorptions. The edge it shares must be between faces $f_i$ and $f_j$. Thus the other face must have label $(i, i, j, 1)$ or $(j, j, i, 1)$ after all absorptions. But in either case this face would be in the same sequence as the first. It follows that an edge cannot be shared by two faces in $F_1 - F_1'$. Thus the claim follows.

It is not hard to verify that the vertices of attachment of any hammock are the only vertices shared with any other hammocks. □

**Lemma 2.2.** Let $\hat{G}_1$ be a neatly prepared embedded undirected planar graph with a face-on-vertex covering of $p' > 1$ faces. There are $\max\{3p' - 6, 1\}$ hammocks in a hammock decomposition of $\hat{G}_1$.

**Proof.** If $p' = 2$, then clearly there is only one hammock. For $p' > 2$, consider the following construction. Generate embedded graph $\hat{G}_h$ from $\hat{G}_1$ as follows. First mimic the absorption of faces by contracting and deleting edges as discussed previously. After the absorption of faces has been mimicked, we compress major hammocks as follows. For

every edge that is on some face $f_i$ in $F_1'$ and in a major hammock, contract the edge, and delete one of the two resulting parallel edges. Such operations should be performed so as to preserve the embedding. Call the resulting graph $\hat{G}_h$. It follows that in $\hat{G}_h$ there is a vertex corresponding to each face in $F_1'$, and each edge in $\hat{G}_h$ corresponds to a hammock in $\hat{G}_1$. It is also clear that there is no face bounded by two parallel edges in $\hat{G}_h$. There are no faces bounded by a single edge, which follows from the way faces labeled by $(i, i, i, 2)$ are absorbed.

Let $V_h$, $E_h$ and $F_h$ be the sets of vertices, edges and faces of $\hat{G}_h$. Since there is no face bounded by a single edge and no face bounded by just two edges, any face in $\hat{G}_h$ must be bounded by three edges. (Note that there are potentially loops in the graph, but these do not individually enclose faces in the embedding, and similarly that there may be two edges with the same endpoints, but these do not alone bound any face.) Thus $|F_h| = 2|E_h|/3$. Since $\hat{G}_h$ is planar, Euler's formula [H] gives $|V_h| - |E_h| + |F_h| = 2$. Combining yields $|E_h| = 3|V_h| - 6$. Since $|V_h| = p'$, and $|E_h|$ is the number of hammocks, the result follows. □

We now discuss how to convert an embedded undirected graph $\hat{G}_0 = (V_0, E_0, F_0)$, with face-on-vertex covering $F'_0$, into an embedded undirected graph $\hat{G}_1 = (V_1, E_1, F_1)$ with face-on-vertex covering $F_1'$, which is neatly prepared. Recall that $\hat{G}_1$ is neatly prepared if $\hat{G}_1$ has no parallel edges, all faces in $F_1 - F_1'$ are bounded by three edges, no pair of faces in $F_1'$ share a vertex, and the boundary of each face is a simple cycle. First consider any vertex $v$ appearing more than once in a clockwise walk around a face. Since there are no loops or parallel edges in the graph, the preceding and

succeeding vertices $u$ and $w$ on the walk are distinct from $v$ and from each other. Vertices $u$ and $w$ are not adjacent, since every path from $u$ to $w$ must necessarily contain $v$. Add an edge from $u$ to $w$. If the face so split was in $F'_0$, replace it with the resulting face that is enclosed by the clockwise walk, but with edges $(u, v)$ and $(v, w)$ replaced by $(u, w)$. Repeat this operation until the boundary of every face is a simple cycle.

Designate as a *shared vertex* any vertex shared by two faces in $F'_0$. Suppose a face $f_i$ in $F'_0$ contains at least four vertices, with at least one shared vertex $v$. Let $u$ and $w$ be the preceding and succeeding vertices in a clockwise walk around $f_i$. Add an edge from $u$ to $w$, and replace $f_i$ in $F'_0$ with the resulting face that is enclosed by the clockwise walk, but with edges $(u, v)$ and $(v, w)$ replaced by $(u, w)$. Repeat this operation until every remaining shared vertex is on a face in $F'_0$ with three vertices.

For any remaining vertex $v$ shared by faces $f_i$ and $f_j$, replace $v$ by vertices $v^i$ and $v^j$ and edge $(v^i, v^j)$. Replace edges $(v, w)$ by $(v^i, w)$ or $(v^j, w)$, so that the clockwise walks around $f_i$ and $f_j$ are the same except with $v$ replaced by $v^i$ and $v^j$, respectively, and planarity is preserved. Finally add edges as necessary to triangulate faces in $F_0 - F'_0$. The resulting undirected graph $\hat{G}$ satisfies the assumptions stated earlier.

We handle an embedded directed graph $\hat{G}$ in the following way. We assume that if both edges $<v, w>$ and $<w, v>$ are in $\hat{G}$, then they together bound a face. To generate undirected graph $\hat{G}_0$, replace each directed edge $<v, w>$ by an undirected edge $(v, w)$, and remove duplicates. Embedded graph $\hat{G}_1$ is generated from $\hat{G}_0$ in the manner discussed above. The hammocks for $\hat{G}_1$ are determined using the main algorithm of this section. Delete any edges from the hammocks which were added in converting $\hat{G}_0$ to

$\hat{G}_1$, noting that any minor hammocks that lose their single edges can be deleted. Replace the remaining edges by the directed edges that they replaced in the conversion from $\hat{G}$ to $\hat{G}_0$. We call the resulting subgraphs *hammocks* of the embedded directed planar graph.

**Theorem 2.1.** Let $\hat{G}$ be an embedding of an $n$-vertex directed planar graph with a face-on-vertex covering $F'$ of $p'$ faces. Given $\hat{G}$ and $F'$, the above algorithm will generate a decomposition of $\hat{G}$ into $O(p')$ hammocks in $O(n)$ time.

**Proof.** It is reasonable to assume that $\hat{G}$ is presented so that edges are maintained in circular doubly-linked lists in order around each vertex, and are also maintained in circular doubly-linked lists in order around each face. Then the conversions of $\hat{G}$ to $\hat{G}_0$, and $\hat{G}_0$ to $\hat{G}_1$ will take $O(n)$ time. The conversions will result in a planar undirected graph of $O(n)$ vertices, with a face-on-vertex covering of $p'$ faces. The hammocks are determined by our procedure in $O(n)$ time, since each absorption can be performed in constant time, and the handling of a maximal sequence of faces can be performed in time proportional to the number of faces in the sequence. By Lemma 2.1, the decomposition generates outerplanar subgraphs, each of which shares at most four vertices. By Lemma 2.2, the number of such graphs generated will be $O(p')$. □

An undirected embedded planar graph is given in Figure 2 for the directed embedded planar graph in Figure 1. Note that vertex 4 was a shared vertex, and an edge (8, 11) was added to remedy this situation. In addition, edges (1, 14), (3, 12), (5, 15), (6, 16) (7, 11), (7, 21), (7, 23), and (11, 16) were added to triangulate faces not in $F'$.

A decomposition for the directed embedded graph of Figure 1 is given in Figure

3, with the attachment vertices shown as emboldened. The generation and the decomposition for the associated undirected embedded graph in Figure 2 has been discussed earlier. Note that both minor hammocks in that decomposition were discarded, since they were induced by edges that had been added. Also note that the hammock spanning vertices {7, 21, 23} has been trimmed to a subgraph containing vertices {21, 23}. This was done because both edges (7, 21) and (7, 23) were added, and when removed they left vertex 7 isolated within the subgraph. The final decomposition is comprised of 7 outerplanar subgraphs.

## 3. Shortest paths in directed outerplanar graphs

In this section we show how to determine in linear time the labels for all edges in a directed outerplanar graph. The major portion of the section assumes that the outerplanar graph has several nice features, while the latter part of the section shows how to deal with an outerplanar graph that does not have these nice features. A key idea used in both is to make use of the natural tree structure of biconnected outerplanar graphs. In both algorithms, sweeps are made through the graph based on this tree structure. Another key idea used in our algorithm is the notion of split vertices, which are actually the initial values in the intervals labeling the edges, and thus are the values stored in the compact routing tables.

We now discuss briefly the organization of the section. We first identify the nice features assumed for most of the section. We then define the notion of a split vertex, and also discuss the natural tree structure of an outerplanar graph. Given this preliminary discussion, we are then able to provide an overview of our algorithm, which employs a

sweep through the tree structure, processing each face in turn. We introduce the data structures upon which our algorithm operates, and then give a detailed discussion about how each face is processed. We then establish the correctness and time complexity of this algorithm. Finally we discuss how to handle an outerplanar graph in which the nice features identified earlier are not present.

We now identify the nice properties that we assume in the major portion of this section. We assume that for each directed edge $<v, w>$ there is an edge $<w, v>$ in the graph, and that the graph with the orientation of edges removed is biconnected. We also assume that edge costs satisfy the *generalized triangle inequality*, i.e., each edge $<v, w>$ is a shortest path from $v$ to $w$. We assume that vertices are named in clockwise order around the exterior face. With the vertices named in this order, we can describe meaningful sets of vertices using interval notation. For example, $\{i+1, i+2, \cdots, j-1\}$ can be described by the open interval $(i, j)$, and $\{i+1, i+2, \cdots, j\}$ by the half-open interval $(i, j]$. At the end of this section we examine how to handle a directed outerplanar graph in which these assumptions are not necessarily satisfied.

We next define several terms, that lead up to the definition of a split vertex. Define an *interior face* in an outerplane embedding to be any face other than the exterior face that is bounded by more than two edges, i.e., not bounded by a pair of edges $<v, w>$ and $<w, v>$ for any $v$ and $w$. Recall that for every edge $<v, w>$ we defined $S(v, w)$ to be the set of vertices such that there is a shortest path from $v$ to each vertex in $S(v, w)$ with the first edge on this path being $<v, w>$. We now specify the tie-breaking rule that guarantees that $u$ will be in one set $S(v, w)$ for each $v$. Let $w$ and $w'$ be neighbors of $v$ on some interior face, with $w$ in the open interval $(v, w')$ and with $u$ in the half-

open interval $(w, w']$. If there is a shortest path from $v$ to $u$ with the first edge on this path being $<v, w>$ and also a shortest path from $v$ to $u$ with the first edge on this path being $<v, w'>$, then $u$ is in only the set $S(v, w')$. If a vertex $u$ is in a set $S(v, w')$, we say that edge $<v, w'>$ *claims* $u$. Let $z$ be the farthest vertex from $v$ in a counterclockwise direction around the exterior face that is claimed by $<v, w'>$. We call $z$ the *split vertex of* vertex $v$ *relative to* neighbors $w$ and $w'$, or the *split vertex* for $(v, w, w')$.

Consider the outerplanar graph in Figure 4. Note that it satisfies the assumptions in the first paragraph of this section. There are four interior faces. Consider the face containing vertices 5, 6, 7, 12, 13, 14, 19 and 20. Vertex 7 has neighbors 12 and 6 on this face. Edge <7,6> claims vertices in the interval [17,7), and edge <7,12> claims vertices in the interval [12,17). The split vertex of 7 relative to neighbors 12 and 6 is vertex 17. (Vertex 7 is also on the face containing vertices 7, 8, $\cdots$, 12. Edge <7,8> claims vertices in [8,12). The split vertex of 7 relative to neighbors 8 and 12 is vertex 12.)

We next discuss the natural tree structure of the outerplanar graph. Consider a relation on interior faces, with two faces related if they are separated by precisely two edges, $<v, w>$ and $<w, v>$, for any $v$ and $w$. There is a natural tree structure based on this relation. (This is the dual graph restricted to interior faces.) Root this tree at an interior face that is related to only one other interior face. Our algorithm sweeps through the tree bottom-up, i.e., an interior face is handled once all interior faces that are children of it in the tree have been handled. For any interior face $f$ other than the root, there is a unique interval $[x, x']$ comprising the set of names of all vertices on faces in the subtree rooted at $f$. Associate with each interior face other than the root the pair of edges $<x, x'>$ and $<x', x>$. Associate with the interior face that is the root a pair of edges

$<x, x'>$ and $<x', x>$ on the exterior face, where $x$ follows $x'$ in clockwise order around the exterior face.

We illustrate the tree with respect to the outerplanar graph in Figure 4. Let the root be the interior face with vertices in [20,5]. Edges <7,12> and <12,7> will be associated with the face containing vertices in [7,12], edges <14,19> and <19,14> will be associated with the face containing vertices in [14,19], and edges <5,20> and <20,5> will be associated with the face containing vertices in $[5,7] \cup [12,14] \cup [19,20]$. We choose edges <3,2> and <2,3> as the edges associated with the root.

As the algorithm sweeps up through the tree, it determines split vertices. The algorithm *processes* each interior face $f$ in turn, by which we mean the following. For face $f$ not the root of the tree, let interval $[x, x']$ be associated with $f$. After face $f$ is processed, the split vertex $z$ will have been found for each triple $(v, w, w')$ such that $z$ is in $(x, x']$. (Some of these split vertices may have been found already when proper descendants of $f$ in the tree were processed.) Every other triple $(v, w, w')$ such that $v$, $w$ and $w'$ are in $[x, x']$ will be stored on a list, ordered by appearance of $v$ on the exterior face. This list will mimic a face in that there are edges between consecutive entries on the list, with edge costs that preserve the difference in distances $d(v, x) - d(v, x')$. In addition, every vertex $v$ in $[x, x']$ that can be a split vertex for some vertex not in $(x, x']$ will also be in a list, ordered by appearance of $v$ on the exterior face, with edge costs that preserve the difference in distances $d(x, v) - d(x', v)$. By carefully traversing and manipulating these lists, split vertices can be found and vertices can be eliminated as candidates for split vertices, in time proportional to the number of changes in the lists.

We now discuss the definition and manipulation of these lists. For each interior face $f$ associate a doubly-linked list $L_1(f)$ of triples $(v, w, w')$, where $v$ is a vertex on $f$, and $w$ and $w'$ are clockwise and counterclockwise neighbors respectively of $v$ on $f$. List $L_1(f)$ is an ordered list of all triples $(v, w, w')$ on face $f$, ordered on vertex $v$ around face $f$ from $x$ to $x'$. Each link in the list will have a cost, representing the distance between the corresponding vertices. Also associate with $f$ a doubly-linked list $L_2(f)$ of vertices from $x$ to $x'$. List $L_2(f)$ is an ordered list of vertices on face $f$, all of which are candidates for being split vertices. Again, links will have costs.

Before $f$ is processed, list $L_1(f)$ will be modified to a list $L_1'(f)$ that holds all triples of $f$ and any triples $(v, w, w')$ of descendants of $f$ in the tree, such that the split vertex for $(v, w, w')$ is not determined before $f$ is processed. Certain link costs in $L_1'(f)$ will represent modified distances, which will be adequate for determining split vertices not already identified. The modified distances are chosen to satisfy several properties such as the generalized triangle inequality and the no negative cycle property. Also before $f$ is processed, list $L_2(f)$ will be modified to a list $L_2'(f)$ of vertices in $[x, x']$, such that if $y$ is not on $L_2'(f)$, then $y$ is not a split vertex for any remaining triples. The upcoming Lemma 3.1 will guarantee that list $L_2(f)$ will be modified correctly.

The lists $L_1(f)$ and $L_2(f)$ together represent a face of the embedded graph. If face $f$ is a leaf, then $L_1'(f) = L_1(f)$ and $L_2'(f) = L_2(f)$. If face $f$ is not a leaf, then once these lists have been modified, the resulting lists $L_1'(f)$ and $L_2'(f)$ do not strictly represent in general a face of a graph, since $L_1'(f)$ may contain a triple $(v, w, w')$, but $L_2'(f)$ does not contain the corresponding vertex $v$, or vice versa. Also, the costs may not correspond from $L_1'(f)$ to $L_2'(f)$. This situation seems to be the result of having a

nonsymmetric cost function. We do not know of a simpler approach that is as efficient and avoids using these modified lists.

To process face $f$ with associated edges $<x, x'>$ and $<x', x>$, do the following. Processing face $f$ will consist of determining the split vertex $z$ for every $(v, w, w')$ on $L_1'(f)$ such that $z$ is in $(x, x']$, and modifying the lists $L_1'(f')$ and $L_2'(f')$, where $f'$ is the parent of $f$ in the tree. A certain prefix of $L_1'(f)$ and a certain suffix of $L_1'(f)$ will together constitute the set of triples $(v, w, w')$ on $L_1'(f)$ such that their split vertices $z$ are in $(x, x']$. In turn we shall discuss handling the first triple in $L_1'(f)$, handling the remaining triples in the prefix of $L_1'(f)$, handling the suffix of $L_1'(f)$, modifying list $L_1'(f')$ of the parent $f'$ of $f$, and modifying list $L_2'(f')$.

We first describe how to handle the prefix of $L_1'(f)$. First determine the split vertex $z$ for the first triple $(v, w, w')$ on list $L_1'(f)$. (Note that $v = x$ and $w' = x'$ for this first triple.) This is accomplished by traversing up $L_2'(f)$ from the other end, starting with $y = x'$, and computing for each vertex $y$ the shortest distances from $v$ to $y$ through $w$ and through $w'$. The split vertex $z$ will be the last vertex encountered on $L_2'(f)$ such that $d(v, w') + d(w', z) \leq d(v, w) + d(w, z)$. Note that $d(v, w') + d(w', z) - (d(v, w) + d(w, z))$ is monotonically nondecreasing as $L_2'(f)$ is traversed, since there are no negative cycles. Save a pointer to the list node containing $z$, the split vertex for $x$, and compute the shortest distance from $x'$ to $z$ as the shortest distance from $x$ to $z$ through $w'$ minus the cost of edge $<x, x'>$.

As an example, consider the face $f$ associated with edges <14,19> and <19,14> in Figure 4. Lists $L_1'(f)$ and $L_2'(f)$ will be the same as lists $L_1(f) = (14,15,19)$,

(15,16,14), (16,17,15), (17,18,16), (18,19,17), (19,14,18) and $L_2(f) = 14, 15, 16, 17, 18, 19$, respectively. The algorithm determines the split vertex for (14,15,19) as 18, since $d(14,19) + d(19,18) = 11$ and $d(14,15) + d(15,18) = 12$ while $d(14,19) + d(19,17) = 15$ and $d(14,15) + d(15,17) = 10$.

Each distance computation will require constant time, if we had initially computed the clockwise and counterclockwise distance around each interior face. For $y = x'$, the shortest distance through $w'$ will be the cost of $<x, x'>$, and the shortest distance through $w$ will be the cost of the clockwise distance around $f$ minus the cost of $<x', x>$. In moving from vertex $y$ to vertex $y'$ on $L_2'(f)$, one value needs to be added to the shortest distance through $w'$, and one value needs to be subtracted from the shortest distance through $w$. The cost of link $<y, y'>$ is added to the distance from $v$ through $w'$, and the cost of link $<y', y>$ is subtracted from the distance from $v$ through $w$.

We next discuss handling the remainder of the prefix of $L_1'(f)$. Having found the split vertex $z$ for $(v, w, w')$, we move down the list $L_1'(f)$ to the next entry $(v', w'', w''')$. By the upcoming Lemma 3.2, we know that split vertex $z'$ for $(v', w'', w''')$ is in $[z, x]$. Compute the shortest distances from $v'$ to $z$ through $w''$ and through $w'''$. Reset $v$, $w$ and $w'$ to $v'$, $w''$ and $w'''$ respectively. Reset $y$ to $z$. While $d(v, w') + d(w', y) > d(v, w) + d(w, y)$ and $y \neq x'$, reset $y$ to be the next vertex back toward $x'$ on $L_2'(f)$. When a split vertex $z$ for $(v, w, w')$ is found, once again move to the next triple $(v', w'', w''')$ on the list $L_1'(f)$ from $x$. If $d(v, w') + d(w', y) > d(v, w) + d(w, y)$ where $y = x'$, then the split vertex for $(v, w, w')$ cannot be found on the face. Save a pointer to the list node for the triple $(v, w, w')$, and compute the shortest distance from $v$ to $x$ through $w'$ as the shortest distance from $v$ to $x'$ through $w'$, minus the cost of $<x, x'>$.

In Figure 4, the split vertex for (15,16,14) will be 19. The split vertex for (16,17,15) cannot be found on the face, since $d(16,15)+d(15,19)=16 > d(16,17)+d(17,19)=11$.

To handle triples in the suffix of $L_1'(f)$, perform the same type of computation as above, but reversing the roles of $x$ and $x'$, and the direction in which the lists are traversed. The test will be slightly different because the tie-breaking between edges is not symmetric. The split vertex $z$ for $(v, w, w')$ will be the last vertex $y$ in the interval $(x, x')$ encountered on $L_2'(f)$ such that $d(v, w')+d(w', y) \leq d(v, w)+d(w, y)$. Note that $y$ is not allowed to be $x$, since if $y=x$ and $v=x'$ then by definition $x$ is claimed by edge $<x', x>$, while if $y=x$ and $v \neq x'$ then $z$ may be outside of face $f$. In Figure 4, the split vertex for (19,14,18) will be 16 and the split vertex for (18,19,17) will be 15. (The split vertex for (17,18,16) will be outside of face $f$, and will turn out to be vertex 7.)

Once split vertices have been found for triples in the prefix and suffix of $L_1'(f)$, lists $L_1'(f)$ and $L_2'(f)$ are trimmed and inserted into $L_1'(f')$ and $L_2'(f')$ respectively, where $f'$ is the interior face that is the parent of $f$. If the trimmed version of $L_1'(f)$ is not empty, then insert the trimmed version of $L_1'(f)$ between $x$ and $x'$ in $L_1'(f')$, and set the costs of the links as follows. Let $(v, w, w')$ and $(v', w'', w''')$ be the first and last triples in the trimmed version of $L_1'(f)$. Using distances with respect to $L_1'(f)$, set $c'(v', x')$ to 0, $c'(x', v')$ to $c(x', x)+d(v', x')-d(v', x)$, $c'(v, x)$ to $d(v, x)-d(v', x')$, and $c'(x, v)$ to $c(x, x')-d(v, v')$. Note that $d(v, v')$ is the distance from $v$ to $x$ through $w$ minus the distance from $v'$ to $x$ through $w''$.

The above operation preserves a number of nice properties. It is shown in the

proof of Theorem 3.1 that the generalized triangle inequality is preserved, and that no negative cycles are introduced. It can be verified that $d'(x, x') = c(x, x')$ and $d'(x', x) = c(x', x)$, where $d'$ is the new distance function for list $L_1'(f')$. Thus for any triple $(v, w, w')$ on $L_1'(f')$, the insertion of the trimmed version of $L_1'(f)$ will be transparent with respect to distances from $v$ to other vertices whose triples are already on $L_1'(f')$. In addition, $d'(u, x') - d'(u, x) = d(u, x') - d(u, x)$, where $u$ is any vertex in the interval $[v, v']$ that is on $L_1'(f)$. Since the split vertex for any vertex $u$ in the trimmed version of $L_1'(f)$ will be in $[x', x]$, the modified distances will not affect any choice of split vertex. Thus the split vertices found for triples in the modified list $L_1'(f')$ will be correct. Also note that the time to modify the list is clearly constant.

With respect to Figure 4, the trimmed version of $L_1'(f)$ will contain the triples (16,17,15), (17,18,16) in that order. These will be inserted between (14,19,13) and (19,20,14) on $L_1'(f')$, and the costs of new links on this list will be set as $c'(17,19) = 0$, $c'(19,17) = 7 + 7 - 10 = 4$, $c'(16,14) = 8 - 7 = 1$, and $c'(14,16) = 8 - 4 = 4$.

List $L_2'(f')$ can be modified similarly. Let $z$ and $z'$ be the split vertices of $x$ and $x'$ respectively. By the upcoming Lemma 3.1, no split vertices yet to be determined will fall in $(x, z') \cup (z, x')$. Furthermore, it can be shown that any edge $<v, w'>$, with $v$ in the interval $(x', x)$ that claims vertex $x'$ will also claim $z$. If $z \neq z'$, let $z''$ be the vertex in interval $[z', z)$ that immediately precedes $z$ on $L_2'(f)$. If $z = z'$, then simply give $x'$ in $L_2'(f')$ a new name of $z$. Otherwise, we trim $L_2'(f)$ so that $z'$ is the first vertex and $z''$ is the last vertex, and insert the trimmed version of $L_2'(f)$ between $x$ and $x'$ in $L_2'(f')$. We give $x'$ in $L_2'(f')$ a new name of $z$, and set the costs of the links as follows. Using dis-

tances with respect to $L_2'(f)$, set $c'(x', z'')$ to 0, $c'(z'', x')$ to $c(x, x') + d(x', z'') - d(x, z') - d(z', z'')$, $c'(x, z')$ to $d(x, z') - d(x', z'')$, and $c'(z', x)$ to $c(x', x) - d(z'', z')$. Note that $d(z'', z')$ is the distance from $x'$ to $z'$ minus the distance from $x'$ to $z''$. It can be verified that $d'(x, x') = c(x, x')$, $d'(x', x) = c(x', x)$, and $d'(x', u) - d'(x, u) = d(x', u) - d(x, u)$, where $d'$ is the new distance function for list $L_2'(f')$, and $u$ is any vertex in the interval $[z', z'']$ that is on $L_2'(f)$.

With respect to Figure 4, the trimmed version of $L_2'(f)$ will contain the vertices 16,17 in that order. These will be inserted between 14 and 19 on $L_2'(f')$, and 19 will be relabeled as 18. The costs of new links on this list will be set as $c'(18,17) = 0$, $c'(17,18) = 8 + 7 - 6 - 4 = 5$, $c'(14,16) = 6 - 7 = -1$, and $c'(16,14) = 7 - 2 = 5$.

Once the interior face at the root is handled, let any remaining triples be assigned the split vertex $x$.

In the next two lemmas, we prove the crucial properties about where split vertices can fall, which allow us to traverse the graph efficiently. The first property allows certain vertices to be ruled out as potential split vertices, on the basis of work already completed. We give an example of the first property before stating it formally. Consider interior edge $<x, x'> = <14,19>$, and the face containing vertices in the interval [14,19]. The split vertex for triple $(x, u, x') = (14,15,19)$ is $z = 18$, and the split vertex for triple $(x', x, u') = (19,14,18)$ is $z' = 16$. Consider a vertex in the interval (19,14), say $v = 6$, with neighbors $w = 7$ and $w' = 5$ on the same face. Then the split vertex for $(v, w, w') = (6,7,5)$ will not be in $(x, z') \cup (z, x') = (14,16) \cup (18,19)$, i.e., will not be vertex 15.

**Lemma 3.1.** Let $G$ be a directed outerplanar graph. Let $x$ and $x'$ be endpoints of an

interior edge. Let $u$ and $u'$ be the neighbors of $x$ and $x'$ respectively that are in the interval $(x, x')$ and are on the same interior face as $x$ and $x'$. Let $z$ be the split vertex for $(x, u, x')$, and $z'$ be the split vertex for $(x', x, u')$. Let $v$ be a vertex in $(x', x)$, with neighbors $w$ and $w'$ on the same face, where $w$ is in interval $(v, w')$. Then the split vertex $y$ for $(v, w, w')$ is not in $(x, z') \cup (z, x')$.

**Proof.** Suppose that $y$ is in $(x, x')$. It follows that

$$d(v, w') + d(w', x') + d(x', x) > d(v, w) + d(w, x) \quad (1)$$

$$d(v, w') + d(w', x') \leq d(v, w) + d(w, x) + d(x, x') \quad (2)$$

Suppose that $y$ were in $(x, z')$. Since $z'$ is the split vertex for $(x', x, u')$,

$$d(x', u') + d(u', y) > d(x', x) + d(x, u) + d(u, y). \quad (3)$$

Since $y$ is the split vertex for $(v, w, w')$,

$$d(v, w') + d(w', x') + d(x', u') + d(u', y) \leq d(v, w) + d(w, x) + d(x, u) + d(u, y). \quad (4)$$

Adding (1) and (3) yields a contradiction to (4). Thus $y$ is not in $(x, z')$.

Suppose that $y$ were in $(z, x')$. Since $z$ is the split vertex for $(x, u, x')$,

$$d(x, x') + d(x', u') + d(u', z) \leq d(x, u) + d(u, z) \quad (5)$$

Since $y$ is the split vertex for $(v, w, w')$, and $y$ is in $(z, x')$,

$$d(v, w') + d(w', x') + d(x', u') + d(u', z) > d(v, w) + d(w, x) + d(x, u) + d(u, z). \quad (6)$$

Adding (2) and (5) yields a contradiction to (6). Thus $y$ is not in $(x', z)$.

The lemma then follows. □

The second property indicates in which direction to look for a split vertex, if we already know the split vertex of a relevant triple. This justifies the correctness of our scan through $L_2'(f)$ as we scan through $L_1'(f)$. Consider two triples $(v, w, w')$ and $(v', w'', w''')$ whose vertices are in faces $f'$ and $f''$, resp., that are contained in the subtree rooted at $f$. Suppose that the split vertices for these triples are not determined before face $f$ is processed. There are two cases that arise. Either $f$ is the lowest common ancestor of $f'$ and $f''$ in the tree, or it isn't.

The simpler example is when $f$ is not the lowest common ancestor of $f'$ and $f''$. Then there is an edge $<t, t'>$ on $f$, such that $v$ and $v'$ are both in the interval $(t, t']$. For example, consider the face $f$ containing the vertices 5, 6, 7, 12, 13, 14, 19 and 20, and consider the triples (16,17,15) and (17,18,16). The vertices in these triples are contained in the same face, containing vertices 14, 15, 16, 17, 18, and 19, and this face is thus the lowest common ancestor. The edge $<t, t'>$ is edge <14,19>. The split vertex for triple (16,17,15) is in the interval (19,14). (In fact, the split vertex for $(v, w, w')$ = (16,17,15) is vertex $z = 6$.) Vertex $v' = 17$ is in the interval $(v, t')$ = (16,19), and the split vertex $z'$ for triple $(v', w'', w''')$ = (17,18,16) is also in the interval (19,14). Then $z'$ is in $[z, t)$ = [6,14). (In fact $z' = 7$.)

The more complicated case to state is when $f$ is the lowest common ancestor of $f'$ and $f''$. Consider the face $f$ containing vertices 5, 6, 7, 12, 13, 14, 19 and 20, and consider triples (13,14,12) and (16,17,15). The vertices in triple (13,14,12) are on face $f$, and the vertices in triple (16,17,15) are on a face that is a descendant of face $f$, so that $f$ is the lowest common ancestor. Consider the edges $<u, u'>$ = <12,13> and $<t, t'>$ = <14,19> on this face. The split vertex $z$ for a triple $(v, w, w')$ = (13,14,12) is in the

interval (13,12). (In fact, $z = 3$.) Vertex $v' = 16$ is in the interval (14,19], and the split vertex $z'$ for triple $(v', w'', w''') = (16,17,15)$ is in the interval (19,14). Then $z'$ is in $[z, t) = [3,14)$. (In fact $z' = 6$.)

**Lemma 3.2.** Let $G$ be a directed outerplanar graph. Let $f$ be an interior face, and let $<u, u'>$ and $<t, t'>$ be edges bounding this face, with $u'$ following $u$ in clockwise order around $f$ and similarly for $t'$ and $t$, where either $u = t$ and $u' = t'$ or $u'$ is in $(u, t]$ and $t$ is in $[u', t')$. Let $v$ be a vertex in the interval $(u, u']$ whose split vertex $z$ for $(v, w, w')$ is in $(u', u)$, where $w$ and $w'$ are neighbors on some interior face, with $w$ in the interval $(v, w')$. Let $v'$ be a vertex in the interval $(t, t'] \cap (v, t']$ whose split vertex $z'$ for $(v', w'', w''')$ is in $(t', t)$, where $w''$ and $w'''$ are neighbors on some interior face, with $w''$ in the interval $(v', w''')$. Then $z'$ is in the interval $[z, t)$.

**Proof.** Suppose that $z'$ were in the interval $(t', z)$. Let $P$ and $P'$ be the shortest paths to $z$ from $v$ through $w$ and $w'$ respectively. Let $P''$ and $P'''$ be the shortest paths to $z'$ from $v'$ through $w''$ and $w'''$ respectively. Let $r_1$ be the nearest vertex from $v$ contained in both $P$ and $P''$, and $r_2$ be the nearest vertex from $v$ contained in both $P'$ and $P'''$. Let $r_3$ be the nearest vertex from $v$ contained in both $P$ and $P'''$. Such a vertex exists, by the following argument. Since $z$ is in $(u', u)$, the portion of $P$ on $f$ will include the clockwise path around $f$ from $u'$ to $t'$. Since $z'$ is in $(t, t')$, the portion of $P'''$ on $f$ will include the counterclockwise path around $f$ from $u'$ to $t'$. These portions clearly share a vertex except when $u = u'$ and $t = t'$. In that case, if $v$ and $v'$ are on the same interior face, then either $P$ contains $v'$ or $P'''$ contains $v$. If $v$ and $v'$ are not on the same interior face, then there is a face $f'$ with all its vertices in the interval $[u, u']$ such that it contains edges $<u'', u'''>$,

$<u''', u''>$, $<t'', t'''>$, and $<t''', t''>$, where $v$ is in $[u'', u''']$ and $v'$ is in $[t'', t''']$. Thus the previous argument applies.

Since the portion of $P'$ from $v$ to $r_2$ is a shortest path,

$$d(v, r_2) \leq d(v, r_3) + d(r_3, r_2) \tag{7}$$

Similarly, since the portion of $P''$ from $v'$ to $r_1$ is a shortest path,

$$d(v', r_1) \leq d(v', r_3) + d(r_3, r_1) \tag{8}$$

Since $z'$ is the split vertex for $(v', w'', w''')$,

$$d(v', r_3) + d(r_3, r_2) + d(r_2, z') \leq d(v', r_1) + d(r_1, z') \tag{9}$$

Since $z$ is the split vertex for $(v, w, w')$, and $z'$ is in $(t', z)$,

$$d(v, r_2) + d(r_2, z') > d(v, r_3) + d(r_3, r_1) + d(r_1, z') \tag{10}$$

Adding (7), (8) and (9) yields a contradiction to (10). Thus $z'$ is not in the interval $(t', z)$. Since $z'$ is by assumption in $(t', t)$, $z'$ is in $[z, t)$. □

**Theorem 3.1.** The above procedure will correctly determine in $O(n)$ time all split vertices in an $n$-vertex directed outerplanar graph in which for every edge $<v, w>$ there is an edge $<w, v>$, the graph with edge orientation removed is biconnected, and the edge costs satisfy the generalized triangle inequality.

**Proof.** Correctness follows from Lemmas 3.1 and 3.2 and the fact that the costs of links of edges $<v', x'>$, $<x', v'>$, $<v, x>$ and $<x, v>$ are set so as to preserve the generalized triangle inequality and the property that there are no negative cycles. We establish the latter fact with respect to the operation of inserting the trimmed version of $L_1'(f)$

between $x$ and $x'$ in $L_1'(f')$. The argument for handling $L_2'(f')$ is similar. We assume inductively that the desired properties hold, and show that the operation of setting the new link costs preserves the properties.

We first derive several useful inequalities. Since the split vertices of $v$ and $v'$ have not been determined by that point in the algorithm, then it must hold that

$$d(v', v) + d(v, x) \leq d(v', x') + c(x', x) \tag{11}$$

and

$$d(v, v') + d(v', x') < d(v, x) + c(x, x') \tag{12}$$

Summing (11) and (12) gives

$$d(v, v') + d(v', v) < c(x, x') + c(x', x) \tag{13}$$

Let $u$ and $u'$ be the first vertices in two triples on the trimmed version of $L_1'(f)$, with the triple containing $u$ preceding the triple containing $u'$. Since there are no negative cycles, $d(v', u') + d(u', v') \geq 0$ and $d(v, u) + d(u, v) \geq 0$. Thus

$$(d(v', u') + d(u', v')) + (d(v, u) + d(u, v)) \geq 0 \tag{14}$$

We now consider the effect on the generalized triangle inequality. Edges whose endpoints are not both in $[x, x']$ will be unaffected. It can easily be verified that the four (new) edges whose costs are set will all satisfy the generalized triangle inequality. We consider the remaining edges. Solving (13) for $d(v', v)$ gives

$$\begin{aligned} d(v', v) &< 0 + c(x, x') + (c(x', x) - d(v, v')) \\ &= c'(v', x') + c(x', x) + c'(x, v) \end{aligned}$$

which is equivalent to

$$(d(v', u') + d(u', u) + d(u, v)) + d(u', v') + d(v, u)$$
$$< d(u', v') + c'(v', x') + c(x', x) + c'(x, v) + d(v, u)$$

Subtracting (14) from the above gives

$$d(u', u) < d(u', v') + c'(v', x') + c(x', x) + c'(x, v) + d(v, u)$$

Thus edge $<u', u>$ is a shortest path from $u'$ to $u$. We derive the similar result for edge $<u, u'>$ as follows. Solving (13) for $d(v, v')$, and adding and subtracting terms, gives

$$d(v, v') < (d(v, x) - d(v', x')) + c(x, x') + ((c(x', x) + d(v', x') - d(v', v) - d(v, x))$$
$$= c'(v, x) + c(x, x') + c'(x', v')$$

where $d(v', x) = d(v', v) + d(v, x)$. Applying (14) gives

$$d(u, u') < d(u, v) + c'(v, x') + c(x, x') + c'(x', v') + d(v', u')$$

We next argue that no negative cycles are created. Since $d'(x, x') = c(x, x')$ and $d'(x', x) = c(x', x)$, no simple negative cycle is introduced that includes both $x$ and $x'$. We next consider the cycle consisting of edges $<v', x'>$ and $<x', v'>$. Now

$$c'(v', x') + c'(x', v') = c(x', x) + d(v', x') - d(v', x)$$

which is greater than 0, by (11). Similarly, for edges $<v, x>$ and $<x, v>$,

$$c'(v, x) + c'(x, v) = d(v, x) - d(v', x') + c(x', x') - d(v, v')$$

$$= d(v, x) + c(x, x') - (d(v, v') + d(v', x'))$$

which is greater than 0, by (12). Thus no negative cycles are introduced.

With respect to trimming $L_2'(f)$, we show that for any vertex $v$ in the interval

$(x', x)$ with neighbors $w$ and $w'$ on the same face, if the split vertex $z'''$ for $(v, w, w')$ is in $(x, x']$, then it is in $(x, z]$. Since $z'''$ is in $(x, x']$

$$c(v, w') + d(w', x') \leq c(v, w) + d(w, x) + c(x, x')$$

By the definition of $z$ as a split vertex,

$$c(x, x') + d(x', z) \leq d(x, z') + d(z', z)$$

Adding the above two inequalities gives

$$c(v, w') + d(w', x') + d(x', z) \leq c(v, w) + d(w, x) + d(x, z') + d(z', z)$$

which establishes that $z'''$ is in $(x, z]$.

We finally analyze the time required by the algorithm. The time to initialize relevant lists is $O(n)$. In handling each interior face, the number of list nodes deleted is within a constant additive term of the number of times that list nodes are examined. Constant work is involved in examining a list node. □

We now consider how to handle an outerplanar graph in which our initial assumptions do not hold. Suppose that outerplanar graph $G$ with edge orientation removed is not biconnected. For every vertex $v$ such that there is no edge $<v, w>$ to the vertex $w$ whose name follows $v$'s name numerically, insert edge $<v, w>$ into $G$ with cost $\infty$. The resulting graph $G$ without edge orientation will clearly be biconnected. Suppose that there is some edge $<v, w>$ in $G$ but no corresponding edge $<w, v>$. For each edge $<v, w>$, if there is no edge $<w, v>$, insert it into $G$ with cost $\infty$. Clearly the only possible change to the edge labeling information resulting from the above operations will be the inclusion into edge labels of vertices that were not previously reachable.

Note that any edges included in the above operations will violate the generalized triangle inequality. We enforce the generalized triangle inequality by identifying any edge that does not satisfy it, labeling the edge as a "pseudo-edge", and changing its cost to be the shortest distance from the vertex representing its tail to the vertex representing its head. We discuss how to do this efficiently in the paragraphs below. Once accomplished, we run our outerplanar algorithm on this modified subgraph. The edge labels which result from this will be in general different from the edge labels for the original subgraph. However, the original edge labels can be recovered by unioning the edge label on each pseudo-edge into the label on the first edge in the shortest path realizing the shortest distance from tail to head, and setting the label on the pseudo-edge to the empty interval. If the shortest distance on the pseudo-edge is realized by two different paths, choose the path that moves counterclockwise around the corresponding face.

We now discuss how to identify edges that violate the generalized triangle inequality and replace them with appropriate pseudo-edges. Recall the relation on interior faces, and the natural tree structure based on this relation. We sweep through the tree structure twice, processing an interior face once on each sweep. On the first sweep, we process an interior face after all its children in the tree have been processed.

An interior face is processed as follows. Determine the cost of the cycles visiting precisely the vertices of the face in clockwise and counterclockwise order. For each edge $<v, w>$ on the clockwise cycle, do the following. If the cost of $<v, w>$ is greater than the cost of the counterclockwise cycle minus the cost of edge $<w, v>$, make $<v, w>$ a pseudo-edge of cost equal to the cost of the counterclockwise cycle minus the cost of $<w, v>$. Perform an analogous operation for counterclockwise edges.

The second sweep processes interior faces in the reverse order from the first sweep, and processes interior faces in the same way.

Suppose that vertices are not named in order around the exterior face. If the names in clockwise order comprise a constant number of consecutive sequences, then the graph can still be handled quickly. Such a case arises with respect to the outerplanar graphs generated in the next section. Rename the vertices in order around the exterior face, apply our outerplanar algorithm to generate edge labels, and then translate the edge labels back to the original names. In the translation, each edge label will grow larger by at most a constant factor.

**Theorem 3.2.** The above algorithm will correctly determine in $O(n)$ time all edge labeling information in an $n$-vertex directed outerplanar graph with real edge costs but no negative cycles.

**Proof.** We first establish the correctness. After a face is processed, the cost of any edge $<v, w>$ on the face represents the shortest distance from $v$ to $w$ along a path constrained to include only vertices that are on the face.

Recall that for each interior face $f$, there is an interval $[x, x']$ comprising the set of vertices on faces in the subtree rooted at $f$, and there is a pair of edges $<x, x'>$ and $<x', x>$ associated with $f$. By induction on the number of faces processed before face $f$ in the first sweep, the following can be established. After face $f$ is processed, the costs on edges $<x, x'>$ and $<x', x>$ represent the shortest distances from $x$ to $x'$ and from $x'$ to $x$ along paths constrained to include only vertices that are in interval $[x, x']$.

By induction on the number of faces processed before face $f$ in the second sweep, the following can then be established. Just before an interior face $f$ is processed in the second sweep, the costs on edges $<x, x'>$ and $<x', x>$ represent unconstrained shortest distances from $x$ to $x'$ and from $x'$ to $x$. Then after processing $f$ on the second sweep, the cost on any edge $<v, w>$ between vertices $v$ and $w$ on $f$ will represent the unconstrained shortest distance from $v$ and $w$. This follows since the processing on the first sweep guarantees that the shortest path from $v$ to $w$ need not detour off of $f$ onto faces that are proper descendants of $f$, and the processing from the second sweep before $f$ is processed guarantees that the shortest path from $v$ to $w$ need not detour off of $f$ onto the parent of $f$.

The time bound follows since the additional time to enforce the triangle inequality, and the time to combine shortest path information from biconnected subgraphs with shortest path information from the rest of the graph, will be $O(n)$. □

**Corollary 3.1.** A shortest path tree rooted at any vertex $v$ in an outerplanar graph can be found in $O(n)$ time.

**Proof.** Reverse the direction of every edge, and apply the above algorithm. For each vertex $w$, put edge $<u, w>$ in the tree if $v$ is in the interval labeling edge $<w, u>$ at vertex $u$ for the reversed graph. □

## 4. Overview of basic approach for planar graphs that are not outerplanar

In this section we first sketch our approach to solving problem 2, i.e., solving all pairs shortest paths in planar graphs, given a good embedding and a good face-on-vertex covering. We then discuss two crucial features of this solution. The first is how to

compress a hammock down to a graph of constant size, while preserving both planarity and the distances between the vertices of attachment in the hammock. The second is the description of a monotonicity property, and its application to the traversal of special subgraphs that arise in section 6.

We now sketch our basic approach, assuming we are given a good embedding and a good face-on-vertex covering. First, we name the vertices, using the following rule:

> *Vertex Naming Rule*: Given an embedded directed planar graph $\hat{G}$ and a face-on-vertex covering $F'$ of cardinality $p'$, vertices are named in clockwise order around each face of $F'$ in turn. If a vertex is encountered more than once in traversing the faces of $F'$, the vertex receives its name on the first encounter.

Second, we determine a hammock decomposition of $\hat{G}$, as discussed in section 2. Third, we find all pairs shortest paths between every pair of attachment vertices. For efficiency, we do this on a compressed graph that we generate as follows. For each major hammock $H$, a compressed version $C(H)$ of $H$ is generated, as described later in this section. Each $C(H)$ will be planar and of constant size. The compressed version $C(G)$ of $G$ is then generated from the compressed versions of the hammocks by identifying corresponding attachment vertices, and adding in the minor hammocks. Compressed graph $C(G)$ will be of size $O(p')$. We then use the all pairs shortest path algorithm for planar graphs in [Fs2] to determine shortest distances between all attachment vertices. This will take $O((p')^2)$ time.

Fourth, for each pair of proper hammocks, we determine succinct shortest path

information for each vertex in one hammock to all vertices in the other hammock. We will show how to do this in time proportional to the total size of both hammocks, which over all pairs of hammocks will be $O(p'n)$, as discussed in section 6.

Fifth, for each hammock, we determine shortest path information between vertices in the same hammock. This would seem to be easy, since we have a linear time algorithm to find shortest path information in outerplanar graphs. However a shortest path in the graph between any pair of vertices in the same hammock $H$ may leave $H$ at one attachment vertex and reenter at another. We give a fast method for determining for each vertex $v$ in $H$ the set of vertices $u$ in $H$ for which the shortest path from $v$ to $u$ stays in $H$. We then determine shortest path information for those paths that leave $H$ by taking two copies of $H$ and treating them as a pair of different hammocks, to which the methods of section 5 are applied. Combining shortest path information within the hammock with shortest path information that detours out of the hammock gives the desired information. All of this can be accomplished in $O(n)$ time, as discussed in section 6.

This completes the sketch of our approach for solving problem 2. The activity in section 5 is seen to dominate the running time of our algorithm, which is $O(p'n)$.

We next discuss how to generate a compressed version $C(H)$ of $H$ for any hammock $H$. The basic idea is to form a subgraph of the hammock that contains shortest paths between pairs of attachment vertices. Then replacement rules are applied to this subgraph which iteratively reduce the number of edges. We first describe how to form the subgraph $B(H)$. Let $a_1$ and $a_2$ be the attachment vertices of $H$ on face $f_i$, and $a_3$ and $a_4$ the attachment vertices of $H$ on face $f_j$, where $a_3$ is adjacent to $a_2$, and $a_1$ is adjacent

to $a_4$, in the corresponding triangulation used to generate the hammocks. Let $T_1$ be a tree formed by taking the union of the shortest paths in $H$ (if they exist) from $a_1$ to $a_2$ and from $a_1$ to $a_3$. Let $T_4$ be a tree formed by taking the union of the shortest paths in $H$ (if they exist) from $a_4$ to $a_2$ and $a_3$, using edges from $T_1$ to break ties. We can use our all pairs shortest paths algorithm for outerplanar graphs to identify these trees in time proportional to the size of the hammock. If we use the same edge labeling information to set up $T_4$ as to set up $T_1$, this tie-breaking will be enforced. Similarly, let $T_2$ be a tree formed by taking the union of the shortest paths in $H$ (if they exist) from $a_2$ to $a_1$ and $a_4$, and $T_3$ be a tree formed by taking the union of the shortest paths in $H$ (if they exist) from $a_3$ to $a_1$ and $a_4$. We initialize a graph $B(H)$ to be the graph $T_1 \cup T_2 \cup T_3 \cup T_4 \cup \{<a_1, a_4>, <a_4, a_1>, <a_2, a_3>, <a_3, a_2>\}$, where the latter edges have cost equal to the shortest distance between their endpoints in $H$. For each such pseudo-edge we associate with it the actual edge in the corresponding shortest path. For edges in each $T_i$, we associate the edge with itself.

We next describe how to repeatedly replace edges and delete vertices until we have compressed $B(H)$ as much as possible, yielding $C(H)$. Temporarily label edges in $T_1 \cup T_4$ as blue, edges in $T_2 \cup T_3$ as red, and the remaining four edges as black. If any edge in $T_1 \cup T_2 \cup T_3 \cup T_4$ is identical to a black edge except for color, then delete it. Recall the definition of an interior face from the previous section. Perform the following operations until they can no longer be applied. Suppose that $<u, v>$ and $<v, w>$ are the only two blue edges incident with vertex $v$, and $u$, $v$, $w$ are consecutive vertices on the same interior face of the current $B(H)$. Then replace $<u, v>$ and $<v, w>$ with blue edge $<u, w>$ of cost $c(u, v) + c(v, w)$. Associate with edge $<u, w>$

the edge associated with $<u, v>$. If $v$ becomes isolated by this operation, then delete it. There is a corresponding operation for red edges. Each such test and replacement can be performed in constant time, if $B(H)$ is stored in the following form. Keep the edges both to and from a vertex on the adjacency list of the vertex. Maintain each adjacency list with edges in clockwise order around the corresponding vertex. If both $<v, w>$ and $<w, v>$ are present in $B(H)$, order edge $<v, w>$ clockwise before $<w, v>$ in $v$'s list, and $<w, v>$ clockwise before $<v, w>$ in $w$'s list.

When no further compression can be done on $B(H)$, remove edge colors, and call the result $C(H)$.

**Lemma 4.1.** Let $H$ be a hammock in a planar graph, with attachment vertices $a_1$, $a_2$, $a_3$, and $a_4$. Graph $C(H)$ is an outerplanar graph of constant size. For any pair of attachment vertices $a_i$ and $a_j$, if there is a path from $a_i$ to $a_j$ in $H$, there is one in the compressed graph $C(H)$, and the lengths of the shortest such paths are identical.

**Proof.** The initial version of the graph $B(H)$ is outerplanar, and each application of an operation leaves $B(H)$ outerplanar. Furthermore, an application of an operation will not change the shortest distance between any pair of attachment vertices.

We argue that the resulting graph $C(H)$ is of constant size as follows. For each attachment vertex $a_i$ of $H$, let $v_i$ be the vertex farthest from $a_i$ that is common to the shortest paths from $a_i$ to each of the two leaves in tree $T_i$. Let $w_i$ be the vertex farthest from $a_i$ that is common to the shortest paths from the roots to $a_i$ in the two trees containing $a_i$ as a leaf. Let $V' = \{a_i, v_i, w_i \mid i = 1, 2, 3, 4\}$. Note that if for example $v_1 \neq v_4$, then the shortest paths from $a_1$ to $a_2$ and from $a_4$ to $a_3$ do not share a common vertex. (This is

shown as follows. Suppose the shortest path $P_{12}$ from $a_1$ to $a_2$ shares a common vertex with the shortest path $P_{43}$ from $a_4$ to $a_3$. Let $x$ be the farthest such vertex from $a_1$. By the tie-breaking rule, the shortest path from $a_1$ to $x$ is a subpath of $P_{12}$, and the shortest path from $x$ to $a_3$ is a subpath of $P_{43}$. Thus the shortest path from $a_1$ to $a_3$ follows $P_{12}$ from $a_1$ to $x$, and then follows $P_{43}$ to $a_3$. Similarly it can be shown that the shortest path from $a_4$ to $a_2$ follows $P_{43}$ from $a_4$ to $x$ and then follows $P_{12}$ to $a_2$. Thus $v_1 = v_4 = x$.) We consider several cases.

If $v_1 \neq v_4$ and $v_2 \neq v_3$, then every vertex $v$ not in $V'$ can have the above operations applied twice, once for blue edges and once for red edges. Thus the only vertices in $C(H)$ will be the at most 12 vertices in $V'$. Similarly, if $v_1 \neq v_4$ and $v_2 = v_3$, then the only vertices in $C(H)$ will be the vertices in $V'$, which will number at most 10, since $v_1 = v_4$, which implies $w_3 = w_2$. A corresponding argument applies if the equality and inequality are reversed in the above condition.

If $v_1 = v_4$ and $v_2 = v_3$, then $V'$ will have cardinality at most 8. The only vertices in $C(H)$ will be $V'$, unless the following condition also holds. From faces $f_1$ and $f_2$ bounding hammock $H$ in $\hat{G}$, if $v_1$ and $w_3$ are not on the same face, and $v_3$ and $w_1$ are not on the same face, then the shortest path from $w_3$ to $v_1$ and the shortest path from $w_1$ to $v_3$ will intersect at at least one vertex. Exactly one of these vertices (call it $z$) will be in $C(H)$. Thus in this case, at most 9 vertices will be in $C(H)$. □

In the next two sections we show how to determine shortest path information between vertices in different hammocks, and between vertices in the same hammock. In the remainder of this section we establish a property that will be particularly useful.

First we recall the *quadrangle inequality*, which will be useful in the proof of the property. Let $w_1$, $w_2$, $u_1$ and $u_2$ be vertices in $\hat{G}$. If the shortest path from $w_1$ to $u_1$ intersects the shortest path from $w_2$ to $u_2$, then

$$d(w_1, u_2) + d(w_2, u_1) \leq d(w_1, u_1) + d(w_2, u_2).$$

To show this, let $z$ be a common vertex on the paths. Then

$$d(w_1, z) + d(z, u_1) = d(w_1, u_1)$$
$$d(w_2, z) + d(z, u_2) = d(w_2, u_2)$$

By the triangle inequality

$$d(w_1, u_2) \leq d(w_1, z) + d(z, u_2)$$
$$d(w_2, u_1) \leq d(w_2, z) + d(z, u_1)$$

Summing the above inequalities and equations yields the claimed result.

We now define the following function that that is described by our monotonicity property. Let $x$ and $y$ be vertices in the graph, and $f$ a face in the embedding of the graph. Define

$$h_{xy}(v) = d(v, x) - d(v, y)$$

where $v$ is a vertex on the boundary of face $f$.

**Lemma 4.2.** *(Monotonicity Property)* Let $x$ and $y$ be vertices and $f$ a face in an embedded planar graph $\hat{G}$. Define $h_{xy}(v) = d(v, x) - d(v, y)$. Let $v'$ and $v''$ be vertices at which $h_{xy}(\cdot)$ achieves a minimum and a maximum, respectively, over all vertices on $f$. Then $h_{xy}(\cdot)$ is nondecreasing on the clockwise sequence of vertices of $f$ from $v'$ to $v''$, and nonincreasing on the clockwise sequence of vertices of $f$ from $v''$ to $v'$. If $x$ is on $f$, then

$h_{xy}(\cdot)$ realizes a minimum at $x$, and if $y$ is on $f$, then $h_{xy}(\cdot)$ realizes a maximum at $y$.

**Proof.** We assume that $h_{xy}(v') < h_{xy}(v'')$, since otherwise the lemma holds trivially. Consider a set of all pairs shortest paths for $\hat{G}$ such that if for any pair of paths, and any two vertices $u$ and $w$, if $u$ precedes $w$ in both paths, then the subpaths from $u$ to $w$ are identical. This can be enforced by assigning each edge a unique index, and breaking ties lexicographically.

Consider any vertex $v$ different from $v'$. Suppose that the shortest path from $v$ to $y$ intersects the shortest path from $v'$ to $x$. Then we claim that $h_{xy}(v) = h_{xy}(v')$. By the quadrangle inequality,

$$d(v, x) - d(v, y) \leq d(v', x) - d(v', y)$$

Since $h_{xy}(\cdot)$ realizes a minimum at $v'$,

$$d(v', x) - d(v', y) \leq d(v, x) - d(v, y)$$

which together imply that $h_{xy}(\cdot)$ realizes a minimum at $v$. If the shortest path from $v$ to $x$ intersects the shortest path from $v'$ to $y$, then by a similar reasoning $h_{xy}(v) = h_{xy}(v')$.

Since we assumed that $h_{xy}(v') < h_{xy}(v'')$, by the above we can conclude that the shortest paths from $v'$ to $x$ and $y$ do not intersect shortest paths from $v''$ to $y$ and $x$, respectively.

Now choose a vertex $v_1$ on face $f$ that is different from $v'$ and $v''$, and such that $h_{xy}(v_1) > h_{xy}(v')$. The above implies that shortest paths from $v_1$ to $x$ and $y$ do not intersect shortest paths from $v'$ to $y$ and $x$, respectively. Then either the shortest path from $v_1$ to $y$ intersects the shortest path from $v''$ to $x$, or the shortest path from $v_1$ to $x$ intersects

the shortest path from $v''$ to $y$.

Let $v_2$ be a vertex in the sequence of vertices on face $f$ between $v''$ and $v_1$ that does not contain $v'$. We shall show that $h_{xy}(v_1) \leq h_{xy}(v_2)$. Suppose the shortest path from $v_1$ to $y$ intersects the shortest path from $v''$ to $x$. Then the shortest path from $v_2$ to $x$ must intersect either the shortest path from $v_1$ to $y$ or the shortest path from $v''$ to $x$. If it intersects the shortest path from $v''$ to $x$, then because of the manner in which ties between shortest paths are broken, the rest of the shortest path from $v_2$ to $x$ will follow the rest of the shortest path from $v''$ to $x$, and thus must intersect the shortest path from $v_1$ to $y$ anyway. Let the shortest path from $v_2$ to $x$ intersect the shortest path from $v_1$ to $y$, at vertex $z$. By the quadrangle inequality,

$$d(v_1, x) - d(v_1, y) \leq d(v_2, x) - d(v_2, y)$$

which is the desired result. If the shortest path from $v_1$ to $x$ intersects the shortest path from $v''$ to $y$, then a similar argument establishes that the shortest path from $v_2$ to $y$ must intersect the shortest path from $v_1$ to $x$, leading again to $h_{xy}(v_1) \leq h_{xy}(v_2)$.

If $x$ is on face $f$ a minimum for $h_{xy}(\cdot)$ is achieved at $x$, since

$$h_{xy}(v) = d(v, x) - d(v, y)$$
$$\geq d(v, x) - (d(v, x) + d(x, y)) = -d(x, y) = h_{xy}(x)$$

A similar argument applies if $y$ is on $f$. □

We define a related function as follows.

$$\bar{h}_{xy}(v) = d(x, v) - d(y, v)$$

By reversing the direction of all edges in the graph, and then applying the above lemma,

we note that the function $\overline{h}_{xy}(v)$ also possesses the Monotonicity Property.

Consider a graph in which $p' = 2$, and the two faces in $f_1$ and $f_2$ share two vertices, $x$ and $y$, in common. Consider the two components $C_1$ and $C_2$ separated by the pair $x$ and $y$. Note that each component is outerplanar. We take advantage of the monotonicity property in a procedure *MON_LABEL* that generates labels for routing from one component through $x$ or $y$ to the other component. Let $v$ be in one component, and $u$ in the other. The shortest path from $v$ to $u$ will be through $x$ if

$$d(v, x) + d(x, u) < d(v, y) + d(y, u)$$

which holds if and only if

$$h_{xy}(v) = d(v, x) - d(v, y) < d(y, u) - d(x, u) = \overline{h}_{yx}(u)$$

The basic idea behind the application of the property is to simultaneously walk through one component and through the other component, using the $h_{xy}$ and $\overline{h}_{yx}$ functions, as though one wanted to merge two ordered lists of values.

Shortest paths from vertices in one component to vertices in the another component can be computed in linear time as follows. We first compute the values $h_{xy}(v)$ for all vertices $v$ and return a list of vertices in each component ordered by $h_{xy}(v)$, and do the same for $\overline{h}_{yx}(u)$. This is accomplished by doing the following in each component $C_i$, $i = 1, 2$. Run the outerplanar algorithm from the previous section to generate the edge labels. Determine the shortest path trees rooted at $x$ and $y$ as follows. Temporarily reverse the direction of edges, run the outerplanar algorithm on the result, and then select for each vertex $v \neq x$ the edge $<w, v>$, where $x$ is in the label for edge $<v, w>$ for the reversed graph. Once the shortest path trees have been found, traverse each tree and

store at each vertex $v$ the distances $d(v, x)$ and $d(v, y)$. For each vertex $v$ form the difference $h_{xy}(v) = d(v, x) - d(v, y)$. By the Monotonicity Property, this difference is monotonically nondecreasing as $v$ moves around either face from $x$ to $y$. Merge the list of vertices on each of the faces $f_1$ and $f_2$, in order of nondecreasing value $h_{xy}(v)$, yielding a list $l_i$ of vertices $v$ for component $C_i$. We assume that the first entry on $l_i$ is $x$. Temporarily reverse the direction of all edges in the component, reverse the roles of $x$ and $y$, and repeat the above. The result will be to compute $\overline{h}_{yx}(u) = d(y, u) - d(x, u)$, along with the list $\overline{l}_i$ of vertices $u$, ordered by nondecreasing $\overline{h}_{yx}(u)$.

For each vertex $v$ in component $C_i$, $i = 1, 2$, define $S_x(v)$ to be the set of vertices in component $C_{3-i}$ whose shortest path from $v$ goes through $x$, and $S_y(v)$ to be the set of vertices in $C_{3-i}$ whose shortest path goes through $y$. The set $S_x(v)$ (and also set $S_y(v)$) is the union of two sets of vertices, each set containing consecutive vertices on one of the faces $f_1$ and $f_2$. Assume that the vertices are named according to the vertex naming convention. It follows that each set $S_x(v)$ (and also set $S_y(v)$) can be described as the union of at most four intervals.

We finally describe the simultaneous walk through both components. For $i = 1, 2$, we then examine the vertices $v$ of component $C_i$, in order of nondecreasing value $h_{xy}(v)$, and simultaneously examine the vertices $u$ of component $C_{3-i}$ in order of nondecreasing value $\overline{h}_{yx}(u)$. This is done as follows. Initialize $S_y$ to the empty set, and $S_x$ to be the intervals describing vertices in component $C_{3-i}$. Set $v$ to the first entry on $l_i$, and $u$ to the first entry on $\overline{l}_{3-i}$. While $h_{xy}(v) \geq \overline{h}_{yx}(u)$, delete $u$ from $S_x$, insert $u$ into $S_y$, and reset $u$ to the next vertex on list $\overline{l}_{3-i}$. When $h_{xy}(v) < \overline{h}_{yx}(u)$, set $S_x(v)$ to $S_x$, set $S_y(v)$

to $S_y$, and reset $v$ to the next vertex on list $l_i$. If the sets $S_x(v)$ and $S_y(v)$ are each maintained as the union of two sets of vertices when insertions or deletions are performed, the work performed between consecutive resettings of $u$ and $v$ will be constant. For any vertex $v$ not equal to $x$ or $y$, the edge $<v, w>$ incident from $v$ that contains $x$ in its edge label for its component will receive the set $S_x(v)$ into its label. The edge incident from $v$ that contains $y$ is handled similarly. This completes our description of procedure *MON_LABEL*.

**Lemma 4.3.** Let $\hat{G}$ be an embedded planar graph in which there is face-on-vertex covering of cardinality 2, with its $n$ vertices named according to the vertex naming convention. Suppose these two faces share two vertices that separate $\hat{G}$ into two components. Procedure *MON_LABEL* generates edge labels for any vertex in one component to vertices in the other component in $O(n)$ time.

**Proof.** By Theorem 3.2, the time to generate edge labels within each component is $O(n)$. Note that while vertices around the border of a component (as opposed to a face $f_1$ or $f_2$) are not necessarily named in order, the names in clockwise order comprise a constant number of consecutive sequences. By the remark preceding Theorem 3.2, this involves additional expense of at most a constant multiplicative factor. The time to compute shortest path trees rooted at $x$ and $y$ is $O(n)$, since the time to identify all appropriate edges $<w, v>$ is proportional to the total size of all edge labels, which is $O(n)$. The creation of the lists $l_i$ and $\overline{l_i}$ by merging will take $O(n)$ time, and the routine to search these lists will also take $O(n)$ time. □

## 5. Handling shortest paths between two hammocks

In this section we give an algorithm to generate shortest path information between vertices in two different hammocks, assuming that distances between their vertices of attachment are known. Our approach is based on computing information about certain constrained shortest paths, and then combining it to yield information about less and less constrained shortest paths, culminating with unconstrained shortest paths. We first give a utility routine whose output is used in generating information about highly constrained shortest paths. Then we define information for various levels of constrained shortest paths. Finally, we show how to generate information about less constrained shortest paths, given information about more constrained shortest paths.

We first give a utility routine that produces very basic information. Let $H$ be a hammock with attachment vertices $a_1$, $a_2$, $a_3$, $a_4$, and let $x$ be a vertex not in $H$. Let $N(x, H, a_i)$ be the set of vertices in $H$ such that a vertex $y$ is in $N(x, H, a_i)$ if and only if $y$ is in $H$ and a shortest path from $x$ to $y$ goes through $a_i$, but no shortest path from $x$ to $y$ goes through any $a_j$ for $j < i$. Suppose the shortest distances are given from $x$ to each of $a_1$, $a_2$, $a_3$, and $a_4$. We describe a procedure to determine the sets $N(x, H, a_i)$, for $i = 1, \cdots, 4$. First generate graph $H'$ from $H$ by introducing vertices $x_i$, $i = 1, \cdots, 4$, and edges $<x_i, a_i>$, $i = 1, \cdots, 4$, with cost equal to the shortest distance from $x$ to $a_i$ in the original graph.

Next we label each vertex in $H'$ with the shortest distance to the nearest $x_i$ as follows. First identify shortest path trees in $H'$ rooted at each $x_i$. For each $x_i$, $i = 1, 2, 3, 4$, traverse its shortest path tree, labeling a vertex $v$ with $i$ and the distance from $x_i$ to $v$ if the

distance to $v$ is smaller than its previous distance. Note that the vertices in each $N(x, H, a_i)$ will be the union of two sets of vertices, each set contiguous along one face of the hammock. Since vertices are named in order around each face, $N(x, H, a_i)$ can be described by the union of four intervals. It is easy to traverse the edges along each face bounding the hammock, forming succinct descriptions of the four sets. Call the above procedure *ATTACH_CLAIM*.

**Lemma 5.1.** Given a hammock $H$ with attachment vertices $a_i$, $i = 1, \cdots, 4$, vertex $x$ not in $H$, and shortest distances from $x$ to the $a_i$, procedure *ATTACH_CLAIM* will generate the four sets $N(x, H, a_i)$ in $O(n')$ time, where $n'$ is the size of $H$.

**Proof.** By Corollary 3.1, each of the four shortest path trees can be determined in $O(n')$ time. Traversing the shortest path trees, and then generating the sets $N(x, H, a_i)$, will each take $O(n')$ time. □

As before, let $H$ be a hammock, and $x$ a vertex not in $H$. Let $N^R(x, H, a_i)$ be the set of vertices in $H$ such that a vertex $y$ is in $N^R(x, H, a_i)$ if and only if $y$ is in $H$ and a shortest path from $y$ to $x$ goes through $a_i$, but no shortest path from $y$ to $x$ goes through any $a_j$ for $j < i$. Sets $N^R(x, H, a_i)$ can be computed in a fashion similar to that of $N(x, H, a_i)$, by reversing the direction of every edge. Note that given shortest distances between all attachment vertices in the graph, and shortest distances from any vertex $x$ to the attachment vertices of its hammock, it is easy to compute, in constant time, shortest distances from $x$ to the attachment vertices of any hammock.

We now discuss the information for various types of constrained shortest paths. We start by defining this information from the least constrained to the most constrained.

We present this information in the form of sets. Let $H_1$ and $H_2$ be distinct hammocks. Let $v$ be a vertex in $H_1$, and $<v, w>$ an edge in $H_1$. Let $M_0(v, w, H_2)$ be the set of vertices $u$ in $H_2$ whose shortest paths from $v$ to $u$ include edge $<v, w>$. In defining these sets $M_0$, as in subsequently defining sets $M_1$, $M_2$, and $M_4$, we assume that ties in the lengths of paths are broken in the following way to yield shortest paths. Among various choices of paths of shortest length, the preferred path will go through an attachment vertex of $H_1$ of smallest possible index, and given that through an attachment vertex of $H_2$ of smallest possible index, and given that will be consistent with the shortest path information generated by our outerplanar algorithm within each of $H_1$ and $H_2$.

Let $x_1$ be an attachment vertex of $H_1$. Let $M_1(v, w, H_2, x_1)$ be the set of vertices $u$ in $H_2$ whose shortest paths from $v$ to $u$ go through vertex $x_1$ and include edge $<v, w>$. Clearly $M_0(v, w, H_2)$ is the union of $M_1(v, w, H_2, x_1)$ over all choices of $x_1$.

Let $y_1$ be a second attachment vertex of $H_1$. For vertices $u$ in $H_2$, we term as *type 1 constrained shortest paths* those paths from $v$ to $u$ that are shortest subject to the constraint that they go through either $x_1$ or $y_1$. Let $M_2(v, w, H_2, x_1, y_1)$ be the set of vertices $u$ in $H_2$ whose type 1 constrained shortest paths from $v$ to $u$ include edge $<v, w>$ and vertex $x_1$. A vertex $u$ is in $M_1(v, w, H_2, x_1)$ if for each attachment vertex $y_1 \neq x_1$ of $H_1$, $u$ is in $M_2(v, w, H_2, x_1, y_1)$.

Let $x_2$ and $y_2$ be attachment vertices in $H_2$. Recall that for attachment vertex $y$ in hammock $H$ and vertex $x$ not in $H$, $N(x, H, y)$ is the set of vertices in $H$ whose shortest paths from $x$ go through $y$. For vertices $u$ in $H_2$, we term as *type 2 constrained shortest paths* those paths from $v$ to $u$ that are shortest subject to the constraint that they go

through either both $x_1$ and $x_2$ or both $y_1$ and $y_2$. Let $M_4(v, w, H_2, x_1, y_1, x_2, y_2)$ be the set of vertices $u$ in $N(x_1, H_2, x_2) \cap N(y_1, H_2, y_2)$ whose type 2 constrained shortest paths from $v$ to $u$ include edge $<v, w>$ and vertices $x_1$ and $x_2$. Then $M_2(v, w, H_2, x_1, y_1)$ is the union of $M_4(v, w, H_2, x_1, y_1, x_2, y_2)$ over all choices of $x_2$ and $y_2$ for which $v$ is in $N^R(x_2, H_1, x_1) \cap N^R(y_2, H_1, y_1)$.

We now show how the $M_i$ sets can be generated, once certain $N(\cdot,\cdot,\cdot)$ sets have been computed. We start with information about the most constrained shortest paths, and work toward the least constrained shortest paths. We can generate $M_4(v, w, H_2, x_1, y_1, x_2, y_2)$ as follows. Each set $N(x, H, y)$ can be computed using procedure *ATTACH_CLAIM*. Generate the graph $G(x_1, y_1, x_2, y_2)$ induced on $N(x_1, H_2, x_2) \cup N(y_1, H_2, y_2) \cup N^R(x_2, H_1, x_1) \cup N^R(y_2, H_1, y_1)$, with the edges $<x_1, x_2>$, $<x_2, x_1>$, $<y_1, y_2>$, and $<y_2, y_1>$ added, of cost equal to the length of the corresponding shortest paths. Perform procedure *MON_LABEL* to generate edge labels for this graph. For edge $<v, w>$ incident from vertex $v$ in $N^R(x_2, H_1, x_1) \cap N^R(y_2, H_1, y_1)$, intersect its label with $N(x_1, H_2, x_2) \cap N(y_1, H_2, y_2)$ and with the label on the end of edge $<x_1, x_2>$ incident at $x_1$.

Once all sets $M_4(v, w, H_2, x_1, y_1, x_2, y_2)$ are generated, then set operations can be performed to yield all sets $M_2(v, w, H_2, x_1, y_1)$, then all sets $M_1(v, w, H_2, x_1)$, and finally all sets $M_0(v, w, H_2)$. Each set generated should be represented in the compact interval notation. Once all sets $M_0(v, w, H_2)$ have been computed for all $v$ in $H_1$, a similar computation will yield all sets $M_0(u, w, H_1)$.

**Theorem 5.1.** Let $H_1$ and $H_2$ be hammocks of sizes $n_1$ and $n_2$, resp., in an embedded

graph $\hat{G}$. Given shortest distances between the vertices of attachment of $H_1$ and $H_2$, the above procedure generates edge labels for any vertex in one hammock to any vertex in the other hammock in $O(n_1+n_2)$ time.

**Proof.** We first address the correctness of the set computation. It is clear that $M_0(v, w, H_2)$ is the union of $M_1(v, w, H_2, x_1)$ over all choices of $x_1$. It is also clear that a vertex $u$ is in $M_1(v, w, H_2, x_1)$ if for each attachment vertex $y_1 \neq x_1$ of $H_1$, $u$ is in $M_2(v, w, H_2, x_1, y_1)$.

We next argue that $M_2(v, w, H_2, x_1, y_1)$ is the union of $M_4(v, w, H_2, x_1, y_1, x_2, y_2)$ over all choices of $x_2$ and $y_2$ for which $v$ is in $N^R(x_2, H_1, x_1) \cap N^R(y_2, H_1, y_1)$. Since $v$ is in $N^R(x_2, H_1, x_1)$, the shortest path from $v$ to $x_2$ goes through $x_1$. Since $v$ is in $N^R(y_2, H_1, y_1)$, the shortest path from $v$ to $y_2$ goes through $y_1$. Consider a vertex $u$ in $M_4(v, w, H_2, x_1, y_1, x_2, y_2)$. Since $u$ is in $N(x_1, H_2, x_2)$, if the shortest path from $v$ to $u$ goes through $x_1$, it goes through $x_2$. Since $u$ is in $N(y_1, H_2, y_2)$, if the shortest path from $v$ to $u$ goes through $y_1$, it goes through $y_2$. Thus unioning over $M_4(v, w, H_2, x_1, y_1, x_2, y_2)$ for all choices of $x_2$ and $y_2$ is a correct approach.

The computation of each $M_4$ set is correct, since the graph $G(x_1, y_1, x_2, y_2)$ contains the appropriate vertices.

We next address the time complexity. Given all relevant sets $N(x, H, y)$, each set $M_0(v, w, H_2)$ can be computed in constant time. Each label generated by performing procedure *MON_LABEL* will be the union of at most 6 sets of vertices, each contiguous along one of the faces of hammocks $H_1$ and $H_2$. The computation on graph

$G(x_1, y_1, x_2, y_2)$ need only be performed when $N^R(x_2, H_1, x_1) \cap N^R(y_2, H_1, y_1) \neq \emptyset$ and $N(x_1, H_2, x_2) \cap N(y_1, H_2, y_2) \neq \emptyset$. In this case the vertices in the graph are the union of at most six sets, each contiguous along one of the four faces of the hammocks. Any edge label generated by procedure *MON_LABEL* will be of the same type. The intersection of this label with $N(x_1, H_2, x_2) \cap N(y_1, H_2, y_2)$ will be the union of at most two sets of vertices, each contiguous along a face of $H_2$. The intersection of this result with the label on $<x_1, x_2>$ will yield a set again of the same type. Since a set of contiguous vertices on a face can be described by the union of at most two intervals, $M_4(v, w, H_2, x_1, y_1, x_2, y_2)$ will be at most four intervals.

Each intersection operation involves a constant number of intervals, and hence can be performed in constant time. There are 16 choices of pairs $x_2, y_2$. Unioning the corresponding labels can be done in constant time. There are 3 choices for $y_1$, and the intersection of a constant number of intervals can be done in constant time. For all $v$ in $H_1$, the number of each type of set $M_0, M_1, M_2, M_4$, will be proportional to the total outdegree of $H_1$, or $O(n_1)$. For all $u$ in $H_2$, the number of each type of set will be $O(n_2)$. Thus the total time for the above procedure will be $O(n_1 + n_2)$. □

## 6. Handling shortest paths between vertices in one hammock

In this section we give an algorithm to generate shortest path information between vertices in the same hammock, assuming that distances between its vertices of attachment are known. This problem is easy if shortest paths between vertices in the hammock do not leave the hammock. It is also possible to determine efficiently the information for shortest paths constrained to leave and reenter the hammock, using the methods of the

previous section. The challenging part is determining efficiently for every pair of vertices $v$ and $u$ whether the shortest path from $v$ to $u$ stays in the hammock or leaves and reenters the hammock. Our approach is to determine for each each vertex $v$ the set of vertices of vertices $u$ such that the shortest path from $v$ to $u$ remains in the hammock. The key idea in the appropriate subproblem is to perform a search of the hammock during which we determine the shortest distance between many pairs of selected vertices $v$ and $u$. Using a special-purpose deque allows this to be accomplished in time linear in the size of the hammock.

We first address a simple case in which a shortest path leaves and reenters the hammock $H$, and show that it can be accommodated by a minor modification of the hammock. Suppose the shortest distance from $v$ to $u$ in $G$ is realized by a path $P$ that leaves and reenters $H$ through attachment vertices at the same end, i.e., $P$ would leave and reenter through vertices $a_1$ and $a_4$, or alternatively through $a_2$ and $a_3$. To handle such cases, just augment $H$ with edges $<a_1, a_4>$, $<a_4, a_1>$, $<a_2, a_3>$, and $<a_3, a_2>$ of costs equal to the lengths of the corresponding shortest paths in $G$.

The harder case to handle is when any shortest path $P$ that realizes the shortest distance from $v$ to $u$ leaves $H$ through an attachment vertex at one end and reenters $H$ at the other end. Then a portion of the shortest distance information between vertices in $H$ arises from edge labels for shortest paths within $H$. The rest of the information arises from edge labels for shortest paths between vertices in two copies of $H$. The challenge is to determine when to use each type of information. Our approach is to determine for each vertex $v$ in $H$ a set $U_H(v)$ of vertices $u$ such that the shortest path from $v$ to $u$ is contained in $H$. We shall show that such a set is the union of two sets of vertices, each of

which is contiguous along one of the faces bounding $H$. Thus each set $U_H(v)$ has a constant size description. Then the appropriate portions of edge labels for shortest paths within $H$ can be unioned with the appropriate portion of edge labels arising from shortest paths leaving $H$ and reentering $H$.

We show how to form the sets $U_H(v)$. Let $a_i$ be one attachment vertex of $H$, and $a_j$ be an attachment vertex at the other end of $H$, i.e., $j \neq 5-i$. For each vertex $v$ in $H$ let $U_{Hij}(v)$ be the set of vertices $u$ such that the distance from $v$ to $u$ in $H$ is no longer than the shortest distance from $v$ to $u$ along a path that leaves $H$ at $a_i$ and reenters $H$ at $a_j$. We shall show in Lemma 6.1 that the set $U_{Hij}(v)$ is the union of two sets of vertices, each of which is contiguous along one of the faces bounding $H$. (In fact, each contiguous set of vertices contains, if it is not empty, the attachment vertex on its face at the same end of the hammock as $a_i$.) For each vertex $v$, the set $U_H(v)$ is the intersection of the sets $U_{Hij}(v)$ over all valid choices of $i$ and $j$. Since each of these sets can be described in constant space, the intersection can be formed in constant time.

We describe how to form the sets $U_{Hij}(v)$ for all vertices $v$ in $H$ and for fixed $i$ and $j$. The key insight, which we show in Lemma 6.2, is that if we scan vertices $v$ in order along one face that bounds the hammock, starting from the end of the hammock containing $a_j$, the set $U_{Hij}(v)$ loses vertices in a monotonic fashion. Thus the last vertex in $U_{Hij}(v)$ on each face moves monotonically toward the end of the hammock containing $a_i$ as vertex $v$ moves toward that end along its face. Thus we perform a coordinated search, bringing along enough information to compute shortest distances between $v$ and vertices that are candidates for the last vertex in $U_{Hij}(v)$ on each face. We are able to perform this search in time proportional to the size of $H$ by using a special-purpose deque

and the edge labels to perform a search of $H$.

We now discuss the generation of the sets $U_{Hij}(v)$ in detail. For simplicity of description we assume that $i = 1$ and $j = 3$, i.e., we are considering paths that leave $H$ at $a_1$ and reenter $H$ at $a_3$. Let face $f_1$ be the face containing $a_1$ and $a_2$, and face $f_2$ be the face containing $a_4$ and $a_3$. For any particular choice of $i$ and $j$, $v$ can be on either $f_1$ or $f_2$, and we can determine the vertices of $U_{Hij}(v)$ on either of the faces $f_1$ or $f_2$. The description of our algorithm is instantiated for $v$ on face $f_1$, and for finding the last vertex in $U_{Hij}(v)$ on the face $f_2$. The three other cases can be handled in essentially the same way. We initialize $v$ to $a_2$, and find all pairs shortest distances in $H$ from $v$ to all vertices in $H$. We also initialize $u$ to be the vertex in $U_{Hij}(v)$ closest to $a_3$ on face $f_2$. It takes time proportional to the size of $H$ to find such a vertex, using the shortest path algorithm for outerplanar graphs. As the result of the initialization, we associate the set of vertices from $u$ to $a_4$ on face $f_2$ with vertex $v$. The set should be represented in compact form, as the union of a minimum number of intervals. This form will be of constant size.

Also as a part of the initialization, we set up a deque with heap order [GT] to aid in the search of $H$. The deque will contain the edges in the shortest path from $v$ to $u$, as we move both $v$ and $u$ in $H$. Each edge $<v, w>$ has a cost associated with it, as well as a label $S(v,w)$ in compact form, encoding shortest path information in $H$. One of the implementations of the deque with heap order in [GT] runs in amortized constant time for the deque operations and constant time for the min operation. The deque of [GT] will perform just as well if the min operation is any associative operation with no inverse. We thus define the min operation on two subsets, each described as the union of a minimum number of subintervals, to be the intersection of these subsets, also described

as the union of a minimum number of subintervals. Since the deque represents a shortest path from $v$ to $u$, $u$ must appear in the min taken over all edges in the deque. In addition, we maintain in a straightforward way a value that is the sum of the costs of the edges in the shortest path from $v$ to $u$.

We use the deque in our search as follows, handling in turn each vertex in addition to $v$ on face $f_1$. While $v$ is not $a_1$, we do the following. First, advance from $v$ to the next vertex $v'$ towards $a_1$ on face $f_1$. Second, use edge labels to search from $v'$ towards $u$, stopping when we first encounter a vertex $t$ in the shortest path from $v$ to $u$. Third, we modify the deque. Delete from the deque all edges from $v$ to $t$, in order from the edge incident from $v$ to the edge incident to $t$. Then insert the edges from $v'$ to $t$, in order starting with the edge incident on $t$ and finishing with the edge incident from $v'$. Fourth, set $u'$ to $u$. Fifth, we advance $u'$ as necessary along face $f_2$ towards $a_4$, until $u'$ is in $U_{Hij}(v')$. We discuss in a moment how this advancing operation is done. Sixth, we reset $v$ to $v'$, and $u$ to $u'$, and associate the set of vertices from $u$ to $a_4$ on face $f_2$ with vertex $v$. Once $v = a_1$, the particular case we have described is handled.

We discuss how the advancing operation is performed, in which we advance $u'$ along face $f_2$ as necessary towards $a_4$ until $u'$ is in $U_{Hij}(v')$. While $u'$ is not in $U_{Hij}(v')$, we do the following. First, let $u''$ be the next vertex from $u'$ towards $a_4$ on face $f_2$. Second, while $u''$ is not in the min for the path, delete non-dummy edges from the end of the path. Third, extend the path to $u''$, using the edge labels in $H$ to search for $u''$. Note that the removal of the end of the path, and also the extension of the path can be carried out by deque operations. Fourth, set $u'$ to $u''$. At this point, we are ready once again for while-test involving $u'$. Once $u'$ is in $U_{Hij}(v')$, the advancing operation is complete.

We note that the test to determine if $u'$ is in $U_{Hij}(v')$ can be implemented as follows. For vertices $v$ and $u$ in $H$, let $d_H(v, u)$ be the length of a shortest path from $v$ to $u$ that is constrained to stay in $H$. We must compare $d_H(v', u')$ with $d_H(v', a_1) + d(a_1, a_3) + d_H(a_3, u')$. The value $d_H(v', u')$ will be the total cost of all edges on the path. The values $d_H(v', a_1)$ can be precomputed for all vertices $v'$ in $H$ in time proportional to $H$, and similarly for $d_H(a_3, u')$. The value $d(a_1, a_3)$ is available from the previous all pairs shortest paths computation on the compressed graph $C(G)$. The test itself will take constant time.

We call the above procedure for generating the $U_{Hij}(v)$ sets procedure *UVSEARCH*. The correctness of procedure *UVSEARCH* depends on several properties. Recall the quadrangle inequality: For vertices $w_1$, $w_2$, $u_1$ and $u_2$, if the shortest path from $w_1$ to $u_1$ intersects the shortest path from $w_2$ to $u_2$, then

$$d(w_1, u_2) + d(w_2, u_1) \leq d(w_1, u_1) + d(w_2, u_2)$$

The first property establishes our characterization of $U_{Hij}(v)$.

**Lemma 6.1** Let $H$ be a hammock in graph $G$. Let $x = a_i$ and $y = a_j$ be attachment vertices at opposite ends of $H$. Let $v$ and $u$ be vertices in $H$, with $u$ on face $f$, one of the two faces that bound $H$. If vertex $u$ is in $U_{Hij}(v)$, then so are all vertices on face $f$ from $u$ to the attachment vertex on face $f$ that is at the same end of the hammock as $a_i$.

**Proof.** Let $u'$ be any vertex on face $f$ from $u$ to the attachment vertex on face $f$ at the same end of the hammock as $a_i$. We have two cases. In the first case, if $u$ and $u'$ are on face $f$, and if $v$ comes between $u'$ and $u$ on this face, then the shortest path in $H$ from $v$ to

$x$ intersects the shortest path in $H$ from $y$ to $u'$. Then by the quadrangle inequality,

$$d_H(v, u') + d_H(y, x) \leq d_H(v, x) + d_H(y, u')$$

Since there are no negative cycles in the graph,

$$0 \leq d(x, y) + d_H(y, x)$$

Summing these yields

$$d_H(v, u') \leq d_H(v, x) + d(x, y) + d_H(y, u')$$

which is the desired result.

In the second case, if $u$ and $u'$ are not on face $f$, or they are but $v$ does not come between $u'$ and $u$ on this face, the shortest path in $H$ from $v$ to $u$ intersects the shortest path in $H$ from $y$ to $u'$. By the quadrangle inequality we get

$$d_H(v, u') + d_H(y, u) \leq d_H(v, u) + d_H(y, u')$$

Since $u$ is in $U_{Hij}(v)$,

$$d_H(v, u) \leq d_H(v, x) + d(x, y) + d_H(y, u)$$

Summing these yields

$$d_H(v, u') \leq d_H(v, x) + d(x, y) + d_H(y, u'),$$

which is the desired result. □

The second property establishes our characterization of the monotonic loss of vertices from $U_{Hij}(v)$ as $v$ recedes toward $a_i$.

**Lemma 6.2** Let $H$ be a hammock in graph $G$. Let $x = a_i$ and $y = a_j$ be attachment ver-

tices at opposite ends of $H$. Let $v$ and $u$ be vertices in $H$, with $v$ on face $f$, one of the two faces that bound $H$. If $u$ is not in $U_{Hij}(v)$, then $u$ is also not in $U_{Hij}(v')$ for any vertex $v'$ on face $f$ between $v$ and the attachment vertex on face $f$ that is at the same end of the hammock as $a_i$.

**Proof.** By Lemma 6.1, $u$ cannot be on face $f$ between $v$ and the attachment vertex on this face at the same end of the hammock as $a_i$. Thus the shortest path in $H$ from $v$ to $x$ intersects the shortest path in $H$ from $v'$ to $u$, for any vertex $v'$ on face $f$ between $v$ and the attachment vertex on this face at the same end of the hammock as $a_i$. By the quadrangle inequality we get

$$d_H(v, x) + d_H(v', u) \geq d_H(v, u) + d_H(v', x)$$

Since $u$ is not in $U_{Hij}(v)$,

$$d_H(v, u) > d_H(v, x) + d(x, y) + d_H(y, u)$$

Summing these yields

$$d_H(v', u) > d_H(v', x) + d(x, y) + d_H(y, u),$$

which is the desired result. □

**Lemma 6.3** Let $H$ be a hammock in graph $G$. Let $a_i$ and $a_j$ be attachment vertices at opposite ends of $H$. Procedure *UVSEARCH* forms the sets $U_{Hij}(v)$ for all vertices $v$ in $H$ in time linear in the size of $H$.

**Proof.** We claim that *UVSEARCH* never deletes an edge from the deque and then later reinserts it. This can be seen as follows. Consider three vertices $v$, $v'$ and $v''$ appearing

on the face containing $a_1$ and $a_2$, with $v'$ between $v$ and $a_1$, and $v''$ between $v'$ and $a_1$. Consider three vertices $u$, $u'$ and $u''$ appearing on the face containing $a_4$ and $a_3$, with $u'$ between $u$ and $a_4$, and $u''$ between $u'$ and $a_4$. If the shortest paths from $v$ to $u$ and from $v''$ to $u''$ share an edge, then the use of edge labels ensures that this same edge will be on the shortest path from $v'$ to $u'$.

Thus every edge in the hammock is added to the deque at most once, at either the front or the rear of the path. Every edge in the hammock can be deleted at most once, and will be deleted from either the front or the rear of the path. As discussed already, the deque with heap order structure of [GT] supports amortized constant deque operations, and a constant time for the min operation. Thus inserting and deleting edges will take time proportional to the size of the hammock.

Whenever the cost of the path from $v'$ to $u'$ is accessed, or a min operation is performed, progress is made, either in the form of a deletion from the deque, or advancing $u'$ toward $a_1$. Each such operation can be performed at most once for each edge or vertex in the hammock. Since each such operation takes constant time, the total time for handling such tests will be proportional to the size of the hammock.

The time for performing the searches for $u''$ among the edge labels can be accounted as follows. The vertices along the face containing $a_3$ and $a_4$ are visited in order from $a_3$ to $a_4$. If edges around each vertex are stored in clockwise order according to an outerplane embedding, then the edges can be scanned in order, without backtracking, to find the label containing the current $u''$, starting from the edge whose label contained the previous $u''$. This implies that each interval in an edge label is scanned just a

constant number of times. Thus the time is proportional to the total size of all edge labels, which is proportional to the number of vertices in $H$. □

**Theorem 6.1.** Let $H$ be a hammock of size $n_1$ in an embedded graph $\hat{G}$. Given shortest distances in $G$ between the vertices of attachment of $H$, the above approach generates edge labels for any vertex in $H$ to any other vertex in $H$ in $O(n_1)$ time.

**Proof.** Lemmas 6.1 and 6.2 establish the correctness of our approach for computing the sets $U_{Hij}(v)$. Since the set $U_H(v)$ is the intersection over a number of such sets, $U_H(v)$ is the union of two sets, each of which is a set of contiguous vertices along a face bounding $H$.

Lemma 6.3 shows that the search to determine the sets $U_H(v)$ for all vertices $v$ in $H$ will require time linear in the size of $H$. By Theorem 3.2, determining information for shortest paths constrained to remain in $H$ will take $O(n_1)$ time. By Theorem 5.1, determining information for shortest paths constrained to detour out of $H$ will take $O(n_1)$ time. Combining this information will take constant time per edge in $H$. Thus the total time will be $O(n_1)$. □

We have now given all the pieces of our algorithm as discussed in section 4.

**Theorem 6.2.** Given a planar embedding $\hat{G}$ and a face-on-vertex covering of cardinality $p'$, our algorithms compute all pairs shortest paths in $O(p'n)$ time.

**Proof.** Given the face-on-vertex covering, a hammock decomposition can be determined in $O(n)$ time. The compressed graph $C(G)$ can be determined in $O(n)$ time, and all

pairs shortest paths solved on it in $O((p')^2)$ time, using the algorithm in [Fs2]. Shortest path information between vertices in $H_i$, and all other vertices can be determined in $O(p'n_i + n)$ time, by Theorem 5.1. Summing over all proper hammocks gives $O(p'n)$ time. By Theorem 6.1, shortest path information between vertices in the same hammock $H_i$, of size $n_i$, can be found in $O(n_i)$ time, or $O(n)$ time over all hammocks. □

## 7. Determining an appropriate face-on-vertex covering

In this section we briefly give a solution to problem 3. Given a planar embedding $\hat{G}$ of an undirected planar graph $G$, we show how to generate a face-on-vertex covering whose cardinality is no more than twice the cardinality of a minimum face-on-vertex covering for $\hat{G}$. We find such a covering by using an approximation algorithm based on techniques found in [B]. We first recall several definitions from [B]. A vertex is on *level* 1 if it is on the exterior face in $\hat{G}$. A cycle of level $i$ vertices is called a *level $i$ face* if it is an interior face in the embedded subgraph induced by level $i$ vertices and consistent with the embedding of $\hat{G}$. By an induced embedded subgraph being consistent with $\hat{G}$, we mean that the embedded subgraph can be extended to yield $\hat{G}$ by adding vertices and edges. For each level $i$ face $f$, let $\hat{G}_f$ be the embedded subgraph of $\hat{G}$ induced by all vertices inside $f$ in $\hat{G}$. All vertices on the exterior face of $\hat{G}_f$ are *level* $i+1$ vertices.

We sketch the method of [B], which is a generic approach for approximation algorithms for certain NP-hard problems on planar graphs. The approach guarantees a solution within a fixed degree of closeness to optimal, in time that is the product of $n$ times an exponential in the inverse of the degree of closeness. Let $k$ be a small positive

integer greater than 1, to be chosen subsequently. The idea is to consider $k$ different "partitions" of an embedded planar graph. For each partition, solve a particular hard problem exactly on each subgraph, and union the solutions on the subgraphs together. Then take as the approximate solution the solution to one of the $k$ partitions that is closest in cost to optimal. The exact notion of partition depends on the particular problem being handled. In general a partition is created by repeatedly peeling off vertices in a number of levels to create a subgraph, with every subgraph except the first and the last having exactly $k$ levels, and the first and the last having no more than $k$ levels.

We instantiate this approach for our problem. For $j = 0, 1, \cdots$ and $r = 1, 2, \cdots, k$, let $\hat{G}_{jr}$ be the embedded subgraph of $\hat{G}$ containing every face in $\hat{G}$ incident on a vertex in level $i$, where $k(j-1)+r < i \leq kj+r$. Let all vertices on levels $k(j-1)+r < i \leq kj+r$ be called *required* vertices of $\hat{G}_{jr}$. The general dynamic programming algorithm in [B] can be adapted to find a minimum cardinality subset $F_{jr}$ of faces of $\hat{G}_{jr}$. (We omit the details of this adaptation; it is a relatively straightforward adaptation.) Let $F_r$ be the union of $F_{jr}$ over $j \geq 0$. Choose $F'$ to be a set among $F_r$ of minimum cardinality.

Let the *restricted* face-on-vertex covering problem be the problem of finding a minimum cardinality face-on-vertex covering for an embedded graph in which certain faces are required to be in the covering, other faces are not allowed to be in the covering, and certain vertices are not required to be covered. We note that the above approximation algorithm can easily be modified to yield a restricted face-on-vertex covering of cardinality at most $(k+1)/k$ times the minimum cardinality.

**Lemma 7.1.** Let $\hat{G}$ be an embedded planar graph, and $k > 1$ a positive integer. There is an $O(8^k n)$ time approximation algorithm that generates a restricted face-on-vertex covering for $\hat{G}$ whose cardinality is at most $(k+1)/k$ times the cardinality of a minimum restricted face-on-vertex covering.

**Proof.** For each $r$, the set of embedded subgraphs $\hat{G}_{jr}$ collectively contain all vertices of $\hat{G}$. Thus $F_r$ will be a covering of the required vertices, containing faces required to be in the covering, and excluding faces not allowed to be in the covering.

Consider an optimal face covering $F^*$. In a fashion consistent with [B], we argue that $|F'| \leq |F^*|(k+1)/k$. For $r = 1, 2, \cdots, k$, let $b_r$ be the number of faces in $F^*$ that contain vertices from both levels $kj+r$ and $kj+r+1$ for some $j$. Since $\sum_{r=1}^{k} b_r \leq |F^*|$, there is some $r'$, $1 \leq r' \leq k$, such that $b_{r'} \leq |F^*|/k$. Then a (not necessarily optimal) face covering of $\hat{G}_{jr'}$ will consist of faces in $F^*$ that are incident with required vertices on levels $k(j-1)+r'+1$ through $kj+r'$. Taken over all $j$ the total number of such faces is $|F^*|(1+1/k)$.

The adapted dynamic programming algorithm from [B] will take $O(8^k n_{jr})$ time on graph $\hat{G}_{jr}$, where $n_{jr}$ is the number of vertices in $\hat{G}_{jr}$. For each $r$, the sum of $n_{jr}$ over all $j$ is $O(n)$. Thus the algorithm runs in $O(8^k n)$ time. □

Note that if $|F'| < k$, then $F'$ achieves the minimum.

## 8. Determining an appropriate embedding and an appropriate covering

In this section we address problem 4. Given an undirected planar graph $G$, but no

embedding of $G$, we show how to generate an embedding $\hat{G}$ and a face-on-vertex covering, such that the cardinality of the covering is at most four times the cardinality of a minimum face-on-vertex covering over all possible embeddings. Our approach is based on decomposing $G$ into triconnected components. We initialize a set to hold these components, and then handle elements of the set one at a time recursively. Handling a component corresponds to running the algorithm in the last section in several variations, and based on the relative performance for the variations, choosing a gadget to substitute into a component that shared two vertices with it. The choice of component can encode an ambiguity as to the embedding, which is resolved as the recursion is unwound. Once we have given an algorithm for solving problem 4, we conclude the section by claiming the main result of the paper.

As mentioned in the introduction, there are planar graphs for which one embedding has a face-on-vertex covering of cardinality 2, while another embedding has only face-on-vertex coverings of cardinality $\Theta(n)$. A family of such graphs is represented in Figure 5a, where the number $n$ of vertices is 2 more than a multiple of 3, and $n \geq 11$. The graph can be viewed as consisting of $(n-2)/3$ pieces in the shape of pie slices, with all slices the same, except for the middle one of the three shown. A minimum face-on-vertex covering for this embedding contains $(n-2)/3$ faces. We shall exhibit an embedding of this graph that has a face-on-vertex covering of cardinality 2.

We first discuss the use of triconnected components. We use the linear-time algorithm of [HT1] to decompose $G$ into triconnected components. Each triconnected component will be either a bond, a polygon, or a triconnected graph, and will consist of

actual edges from $G$ and virtual edges representing portions of $G$ that were split off. Any virtual edge in a component will have a corresponding virtual edge in some other component. The decomposition into triconnected components of the graph in Figure 5a is given in Figure 5b. Each virtual edge is drawn as a dashed edge, and placed next to its corresponding edge.

Our algorithm uses a notion of *extended components* of a graph $G$, which are modified triconnected components of $G$. Sets of extended components are defined recursively as follows. The set of triconnected components of planar graph $G$ is a set of extended components of $G$. Let $\Gamma$ be a set of extended components of $G$, with $|\Gamma| > 1$. Then $\Gamma'$ is a set of extended components of $G$, of cardinality $|\Gamma| - 1$, defined as follows. Let $C_1$ be an extended component in $\Gamma$ that contains some number of actual edges and precisely one virtual edge $e$, and let $C_2$ be the extended component in $\Gamma$ that contains the virtual edge $e'$ corresponding to $e$. Then $\Gamma' = \Gamma - \{C_1, C_2\} \cup \{C_3\}$, where $C_3$ is a component generated when edge $e'$ in $C_2$ is replaced by any one of the gadgets in Figure 6. (Our definition allows an arbitrary choice of gadget. Of course our algorithm will make a particular choice of gadget. This choice will be discussed subsequently.) Note that the components in any set of extended components are in one-to-one correspondence with the components in a set of components obtained when certain of the triconnected components of $G$ are merged back together. Thus the components in the set of extended components can be merged together to yield a planar graph $G'$. (Because of the replacement by gadgets, $G'$ will in general be different from $G$.) A set of extended components of the graph in Figure 5a is given in Figure 7a, and a second set is given in Figure 7b.

The designation "can't use" in certain faces of some of the gadgets refers to the

restriction that when a component is embedded in the plane, the corresponding face cannot be used in the face-on-vertex covering. Note that each gadget is symmetrical with respect to reflection about the axis through the top and bottom vertices. Thus for any extended component that was not initially a bond, there are at most two nonequivalent embeddings of this component, one a reflection of the other, in which any one particular face is the exterior face.

We now give a recursive procedure to determine a good embedding and a good face-on-vertex covering. We assume as input a data structure containing a set $\Gamma$ of extended components of $G$, arranged in lists according to the number of virtual edges in each. At the top level of recursion, $\Gamma$ will be the set of triconnected components of $G$. If $\Gamma$ contains exactly one extended component $C_1$, we generate an embedding and a face-on-vertex covering of $C_1$, as discussed below. If $\Gamma$ contains more than one extended component, choose an extended component $C_1$ that has exactly one virtual edge $e$. Remove $C_1$ from the set, handle it, and modify the component $C_2$ that contains the corresponding virtual edge $e'$. Component $C_2$ is modified by replacing virtual edge $e'$ by one of the four gadgets shown in Figure 6, generating a resulting set of extended components $\Gamma'$. We discuss the rule for the choice of gadget below. The algorithm is applied recursively to $\Gamma'$, and the resulting embedding and face-on-vertex covering is finally modified to reflect the processing of $C_1$.

We discuss how to handle the component $C_1$. Suppose $C_1$ was not initially a bond. Choose one of the two embeddings $\hat{C}_1$ of the component, using a linear-time planarity testing algorithm [HT2, ET, BL]. If $C_1$ does not contain a virtual edge, then

use the approximation algorithm from the last section on the embedded component $\hat{C}_1$, using the parameter $k = 4$. If $C_1$ does contain a virtual edge, let $f_1$ and $f_2$ be the faces incident on the virtual edge. Run four versions of the approximation algorithm from the last section on the embedded component $\hat{C}_1$, again using the parameter $k = 4$. In the first version require both $f_1$ and $f_2$ to be used. In the second require $f_1$ to be used and $f_2$ not to be used. In the third require $f_2$ to be used and $f_1$ not to be used. In the fourth require neither to be used. In all four problems, the endpoints of the virtual edge are not required to be covered. Let $p_1, p_2, p_3$ and $p_4$ be the respective number of faces in the face-on-vertex coverings generated. (If no covering is possible given the restrictions, then take the number of faces to be $\infty$.) Without loss of generality, assume $p_2 \leq p_3$.

. If $p_1 \leq \min\{p_2, p_4\}$, then the solution to the first problem is preferred in an approximation, since including $f_1$ and $f_2$ in the covering can only help in covering vertices in other components. Replace the corresponding virtual edge in $C_2$ with the gadget of type 3 in Figure 6. Note that the middle two faces in this gadget are excluded from being used in any face-on-vertex covering of an embedding of $C_2$. This then forces the two faces on either side of this gadget to be in the face-on-vertex covering.

If $p_2 \leq \min\{p_1-1, p_4\}$, then the solution to the second problem is preferred. This follows since including $f_1$ can only help as compared with the solution to the fourth problem, and including $f_2$ later to help cover vertices in other components would boost the total cost only to $p_2+1 \leq p_1$. Replace the corresponding virtual edge in $C_2$ with the gadget of type 2 in Figure 6. This forces one of the two faces on either side of the gadget to be in any face-on-vertex covering.

If $p_4 \leq \min\{p_1-2, p_2-1\}$, then the solution to the fourth problem is preferred. This follows since including one of $f_1$ and $f_2$ later to help cover vertices of other components would boost the total cost only to $p_4+1 \leq p_2$, and including both of $f_1$ and $f_2$ later would boost the cost to $p_4+2 \leq p_1$. In this case, replace the corresponding virtual edge with the gadget of type 1 in Figure 6.

When none of the above conditions hold, we have that $p_1 = p_4+1 \leq p_2$. In this case, we would like to use either the solution to the first or the fourth problem, depending on which is more advantageous. We replace the corresponding virtual edge with the gadget of type 4 in Figure 6. Note that the top and bottom interior faces are not allowed to be used in the face-on-vertex covering. This means that either the middle interior face is used, or both outside faces will be used. Using the outside faces corresponds to choosing the solution to the first problem, while otherwise not using both outside faces corresponds to choosing the solution to the fourth problem. This ambiguity about which solution to use is left unresolved until the procedure returns back from the recursion. This concludes the description of how to handle a component that was not initially a bond.

Suppose component $C_1$ was initially a bond. In the worst case, there will be many different possible embeddings for $C_1$. We describe how to generate an embedding that has a face-on-vertex covering of minimum cardinality, subject to restrictions on how many faces bounding a virtual edge are to be contained in the covering. For convenience, we view any edge in $C_1$ that was an actual edge in the original bond as a gadget of type 1. An embedding will be specified by giving a cyclic ordering, around one of the vertices originally in the bond, of the gadgets in $C_1$, along with the virtual edge, if

present. First have all the gadgets of type 1, then all gadgets of type 4, then one gadget of type 2 if there is one, then all gadgets of type 3, and then the remaining gadgets of type 2. If there is no virtual edge, then the above embedding is sufficient. If there is a virtual edge, then we generate three different embeddings, each dependent on how many faces next to the virtual edge would be required to be used in the covering. If both such faces are, then put the virtual edge after the gadgets of type 3. If exactly one such face is, then put the virtual edge in front of all gadgets. If no faces next to the virtual edge are to be included in the covering, then put the virtual edge after the first gadget in the list. A minimum cardinality face-on-vertex covering of each embedding can be generated by a straightforward greedy algorithm.

**Lemma 8.1.** Let $C_1$ be a component that was originally a bond. Suppose either $C_1$ contains no virtual edge, or $C_1$ contains a virtual edge and an embedding is required to have exactly $i$ faces incident on the virtual edge, where $i$ is 0, 1, or 2. The above algorithm gives an embedding $\hat{C}_1$ that has a minimum cardinality face-on-vertex covering whenever such an embedding exists.

**Proof.** Suppose $C_1$ contains no virtual edge. One face will be needed for each gadget of type 3 or type 4, and every two gadgets of type 2. Furthermore, if there are no gadgets of type 2, and at least one of type 1, then one additional face will be needed. It is easy to verify that the embedding given has a face-on-vertex covering of this size.

When there is a virtual edge in $C_1$, the proof involves verifying a number of cases. Let $g_j$ be the number of gadgets of type $j$ in $C_1$. We note that if $g_2 + g_3 + g_4 > 0$ and $g_1 + g_4 + g_2 < 2$, then there will be no embedding with both of the faces incident on

the virtual edge not used. If either $g_4 = 1$ and $g_1 + g_2 + g_3 = 0$, or $g_3 > 0$ and $g_1 + g_2 + g_4 = 0$, then there will be no embedding with exactly one of those faces not used. □

We complete the discussion of how to handle a component that was initially a bond. If $C_1$ contains no virtual edge, then an embedding and a minimum face-on-vertex covering have been determined. Otherwise, let $p_1$, $p_2$ and $p_4$ be the number of faces in a minimum cardinality covering when respectively 2, 1, 0 faces incident on the virtual edge are included in the covering. Perform the comparisons between $p_1$, $p_2$, and $p_4$ as described earlier, substituting the selected gadget in place of the corresponding virtual edge in some component $C_2$.

We now illustrate how to handle components. Consider the set of triconnected components in Figure 5b. Consider each component that has precisely one virtual edge. For each of these components, the solutions generated by the algorithm of section 7 for the various problems will have $p_1 = 2$, $p_2 = p_3 = 1$, and $p_4$ undefined. Since $p_2 \leq p_1 - 1$, each such component will be replaced by a gadget of type 2. The resulting set of extended components is shown in Figure 7a. The solutions generated by the algorithm of section 7 for the component shown in the middle slice of Figure 7a for the various problems will have $p_1 = 2$, $p_2 = p_3 = 2$, and $p_4 = 1$. Since $p_1 = p_4 + 1 \leq p_2$, the component will be replaced by a gadget of type 4. The solutions generated by the algorithm of section 7 for the components shown in the other slices of Figure 7a for the various problems will have $p_1 = 2$, $p_2 = p_3 = 1$, and $p_4$ undefined. Thus these components will be replaced by gadgets of type 2. The resulting single component is shown in Figure 7b. The face-on-vertex covering generated by the approximation algorithm in section 7 is

shown in Figure 8a. Note that the cardinality of the covering is the smallest possible. This must necessarily be so since the cardinality of this covering is less than $k$.

We finally discuss how to return from the recursion, and resolve ambiguous choices for the embedding. Thus we discuss how to modify the solution for $\Gamma'$ to yield a solution for $\Gamma$. The embedded graph $\hat{G}'$ for $\Gamma'$ contains the gadget that was substituted in place of component $C_1$. Replace the gadget with component $C_1$ minus its virtual edge. If the gadget is of type 2, choose the reflection of $C_1$ that forced the choice of the gadget originally. Union the covering of $C_1$ into the covering for the embedding being constructed. If the gadget is of type 4, choose the covering of $C_1$ that is consistent with the way the vertices of the gadget were covered.

**Theorem 8.1.** The above approximation algorithm generates an embedding of $G$ and a face-on-vertex covering of cardinality at most four times the cardinality of the minimum covering over all possible embeddings.

**Proof.** The proof is by induction on the number of extended components of $G$. Suppose there is just one extended component. If the component was not initially a bond, then there are just two embeddings, which are reflections of each other. Our algorithm uses the approximation algorithm from section 6, that is guaranteed to get within a factor of $(k+1)/k = 5/4$. If the component was initially a bond, then by Lemma 8.1 our algorithm identifies an embedding that allows for a minimum face-on-vertex covering, and finds this covering.

Suppose there is more than one extended component. We assume as the induc-

tion hypothesis that our algorithm gets within a factor of 4 on any graph with fewer extended components. Given a graph $G$, let $p(G)$ be the minimum number of faces in any face-on-vertex covering of any embedding. Let $F^*$ be a face-on-vertex covering for $G$ that is of cardinality $p(G)$. For extended component $C_1$ of $G$ that has one virtual edge, let $F^{*\prime}$ be a minimum cardinality subset of $F^*$ needed to cover all vertices in $C_1$ except the endpoints of the virtual edge. Let $p(C_1)$ be the cardinality of $F^{*\prime}$.

Let $p'$ be the minimum of the $p_i$, $i = 1, 2, 3, 4$, for component $C_1$. Let $G'$ be the graph resulting after our algorithm deletes component $C_1$ and substitutes a gadget in place of a virtual edge in component $C_2$. Let $\hat{p}(G)$ be the cardinality of a covering generated by our algorithm.

Suppose $p(C) \leq 3$. This means that $\min\{p_1, p_2, p_4\} \leq 3$. Consider the case in which $p_1 \leq \min\{p_2, p_4\}$. We have $p(G') \leq p(G) - (p_1 - 2)$, since $p_1$ is the minimum. Thus $\hat{p}(G) \leq p_1 - 2 + \hat{p}(G') \leq p_1 - 2 + 4p(G')$, by the induction hypothesis. Substituting, we get $\hat{p}(G) \leq p_1 - 2 + 4(p(G) - p_1 - 2) \leq 4p(G)$, since $p_1 \geq 2$. Consider the case in which $p_2 \leq \min\{p_1 - 1, p_4\}$. We have $p(G') \leq p(G) - (p_2 - 1)$, since $p_2$ is the minimum, and $p_1 - 2 \geq p_2 - 1$. Thus $\hat{p}(G) \leq p_2 - 1 + \hat{p}(G') \leq p_2 - 1 + 4p(G')$, by the induction hypothesis. Substituting, we get $\hat{p}(G) \leq p_2 - 1 + 4(p(G) - p_2 - 1) \leq 4p(G)$, since $p_2 \geq 1$. Consider the case in which $p_4 \leq \min\{p_1 - 2, p_2 - 1\}$. If $p_4 \leq 2$, then $p(G') \leq p(G) - p_4$, since $p_1 - 2 \geq p_4$ and $p_2 - 1 \geq p_4$, and $p_1$ and $p_2$ are the smallest possible values, not approximations. Thus $\hat{p}(G) \leq p_4 + \hat{p}(G') \leq p_4 + 4p(G')$, by the induction hypothesis. Substituting, we get $\hat{p}(G) \leq p_4 + 4(p(G) - p_4) \leq 4p(G)$. If $p_4 = 3$, then $p(G') \leq p(G) - p_4 + 1$, since $p_4 - 1 \leq p_1 - 2$. Thus $\hat{p}(G) \leq p_4 + \hat{p}(G') \leq p_4 + 4p(G')$, by the

induction hypothesis. Substituting, we get $\hat{p}(G) \leq p_4 + 4(p(G) - p_4 + 1)$ $= 4p(G) - 3p_4 + 4 < 4p(G)$.

Suppose $p(C) > 3$. Since at most two faces in $F^{*\prime}$ cover vertices not in $C$, $p(G') \leq p(G) - p(C) + 2$. Then $\hat{p}(G) \leq p' + \hat{p}(G') \leq p' + 4p(G')$, by the induction hypothesis. Substituting, we get $\hat{p}(G) \leq (5/4)p(C) + 4(p(G) - p(C) + 2) = 4p(G) - (11/4)p(C) + 8$ $\leq 4p(G)$. □

We suspect that the constant of 4 can be improved by careful analysis. We complete our running example by seeing how the embedding and face-on-vertex covering in Figure 8a is expanded to yield a good embedding and face-on-vertex covering for the graph in Figure 5a. For each gadget of type 2 substituted into Figure 7a to yield Figure 7b, we replace the gadget by its corresponding component. Note that we are careful to use the appropriate reflection of components replacing gadgets of type 2. Also note that since the gadget of type 4 had both outside faces in the covering, both outside faces of the component are included. Since these components are rather simple, no faces other than outside faces were in their coverings. The resulting graph, embedding, and covering are shown in Figure 8b. Substituting for gadgets in Figure 8b gives the original graph of Figure 5a, along with an embedding and a face-on-vertex covering. The covering is of cardinality 2, the best possible for this family of graphs.

We are now able to claim the main result of the paper.

**Theorem 8.2.** Let $G$ be a directed planar graph, with $n$ vertices, and real-valued edge costs but no negative cycles. Let $p$ be the minimum cardinality of a face-on-vertex covering over all planar embeddings of $G$. Our algorithm constructs compacted routing

tables for all pairs shortest paths in $G$ in $O(pn)$ time.

**Proof.** The result follows directly from Theorems 6.2 and 8.1 □

## 9. Verifying the triangle inequality, and another encoding

It is possible to determine all edges violating the triangle inequality in time that is better than $O(pn)$ whenever $p$ is $o(n)$. The time will be $O(n+p^2)$, as we now show. Perform all the portions of our algorithm except for finding edge labels between vertices in different hammocks. For each edge $<v, w>$, test if $w$ is in the interval labeling edge $<v, w>$ in the hammock containing $<v, w>$. An edge $<v, w>$ violates the triangle inequality if and only if the test fails.

**Theorem 9.1.** Let $G$ be a directed planar graph, with $n$ vertices, and real-valued edge costs but no negative cycles. Let $p$ be the minimum cardinality of a face-on-vertex covering over all planar embeddings of $G$. All edges that violate the generalized triangle inequality can be determined in $O(n+p^2)$ time.

**Proof.** Since each edge is in some hammock, it is not necessary to find shortest path information for two vertices in different hammocks. The time to perform all the portions of our algorithm except for finding edge labels between vertices in different hammocks is $O(n+p^2)$. Given the edge label information within hammocks, the time to perform each test is constant per edge, or $O(n)$ overall. □

We know of no class of graphs for which the current best algorithm for verifying the generalized triangle inequality is faster than the current best algorithm for solving all

pairs shortest paths. The class of planar graphs with a minimum cardinality face-on-vertex covering of size $p$ appears to be no different, if we allow an alternative encoding of all pairs shortest paths information.

The encoding consists of all pairs shortest distances and shortest paths in the compressed graph $C(G)$, edge labels in each hammock $H$, the sets $U_H(v)$ for all vertices $v$ in each hammock $H$, and shortest distances between each vertex in $H$ to the attachment vertices of $H$.

Given this encoding, the first edge $<v, w>$ on a shortest path from $v$ to $u$ is determined as follows. Suppose $v$ and $u$ are in the same hammock $H$. If $u$ is in $U_H(v)$, then the shortest path from $v$ to $u$ stays within $H$. Thus $<v, w>$ is the edge incident from $v$ with $u$ in its edge label. If $u$ is not in $U_H(v)$, or if $v$ is in hammock $H$ and $u$ is in hammock $H' \neq H$, then we consult the distance information. (If $v$ and $u$ are in the same hammock $H$, but $u$ is not in $U_H(v)$, then let $H' = H$ in the following.) Let $a_1$, $a_2$, $a_3$ and $a_4$ be the attachment vertices of $H$, and $b_1$, $b_2$, $b_3$ and $b_4$ be the attachment vertices of $H'$. Choose $i$ and $j$ to minimize $d(v, a_i) + d(a_i, b_j) + d(b_j, u)$. If $v \neq a_i$, then edge $<v, w>$ will be the edge incident from $v$ with $a_i$ in its edge label. If $v = a_i$, then $<v, w>$ will be the first edge in a shortest path from $a_i$ to $b_j$ in $C(G)$.

**Theorem 9.2.** Let $G$ be a directed planar graph, with $n$ vertices, and real-valued edge costs but no negative cycles. Let $p$ be the minimum cardinality of a face-on-vertex covering over all planar embeddings of $G$. The above encoding of all pairs shortest path information can be computed in $O(n + p^2)$ time.

**Proof.** The above encoding can be generated by performing all the portions of our

algorithm except for finding edge labels between vertices in different hammocks. This requires time $O(n + p^2)$. □

While this encoding can be generated in general more quickly than compact routing tables, it obviously cannot be used in place of compact routing tables for point-to-point message routing in a network.

**Acknowledgement.** The author would like to thank the referees for their careful reading of the paper and their many helpful suggestions.

## References

[AHU] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley (1974).

[B] B. S. Baker, Approximation algorithms for NP-complete problems on planar graphs (preliminary version), *Proc. 24th IEEE Symp. on Foundations of Computer Science*, Tucson (1983) 265-273.

[BM] D. Bienstock and C. L. Monma, On the complexity of covering vertices by faces in a planar graph, *SIAM J. Computing 17*, (1988) 53-76.

[BL] K. S. Booth and G. S. Lueker, Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, *J. Computer and Systems Sciences 13* (1976) 335-379.

[DP] N. Deo and C. Pang, Shortest-path algorithms: taxonomy and annotation, *Networks 14* (1984) 275-323.

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

[ET] S. Even and R. E. Tarjan, Computing an st-numbering, *Theor. Computer Science 2* (1976) 339-344.

[FL] M. R. Fellows and M. A. Langston, Nonconstructive advances in polynomial-time complexity, *Info. Proc. Lett. 26* (1987-88) 157-162.

[Fl] R. W. Floyd, Algorithm 97: shortest path, *Comm. ACM 5* (1962) 345.

[Fs1] G. N. Frederickson, Implicit data structures for the dictionary problem, *J. ACM 30* (1983) 80-94.

[Fs2] G. N. Frederickson, Fast algorithms for shortest paths in planar graphs, with applications, *SIAM J. on Computing 16* (1987) 1004-1022.

[Fs3] G. N. Frederickson, A new approach to all pairs shortest paths in planar graphs, *Proc. 19th ACM Symposium on Theory of Computing*, New York City (May 1987) 19-28.

[FJ1] G. N. Frederickson and R. Janardan, Designing networks with compact routing tables, *Algorithmica 3* (1988) 171-190.

[FJ2] G. N. Frederickson and R. Janardan, Efficient message routing in planar networks, *SIAM J. on Computing 18* (1989) 843-857.

[Fm] M. L. Fredman, New bounds on the complexity of the shortest path problem, *SIAM J. on Computing 5* (1976) 83-89.

[FT] M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. ACM 34* (1987) 596-615.

[GT] H. Gajewska and R. E. Tarjan, Deques with heap order, *Info. Proc. Lett. 22* (1986) 197-200.

[H] F. Harary, Graph Theory, Addison-Wesley, Reading MA, 1969.

[HT1] J. E. Hopcroft and R. E. Tarjan, Dividing a graph into triconnected components, *SIAM J. Computing 2* (1973) 135-158.

[HT2] J. E. Hopcroft and R. E. Tarjan, Efficient planarity testing, *J.ACM 21* (1974) 549-568.

[MS] J. I. Munro and H. Suwanda, Implicit data structures for fast search and update, *J. Computer and System Sciences 21* (1980) 236-250.

[SK] N. Santoro and R. Khatib, Labelling and implicit routing in networks, *Computer Journal 28*, (1985) 5-8.

[vLT] J. van Leeuwen and R. B. Tan, Computer networks with compact routing tables, in *The Book of L*, G. Rozenberg and A. Salomaa (eds.), Springer-Verlag, New York (1986) 259-273.

[W] S. Warshall, A theorem on Boolean matrices, *J. ACM 9* (1962) 11-12.

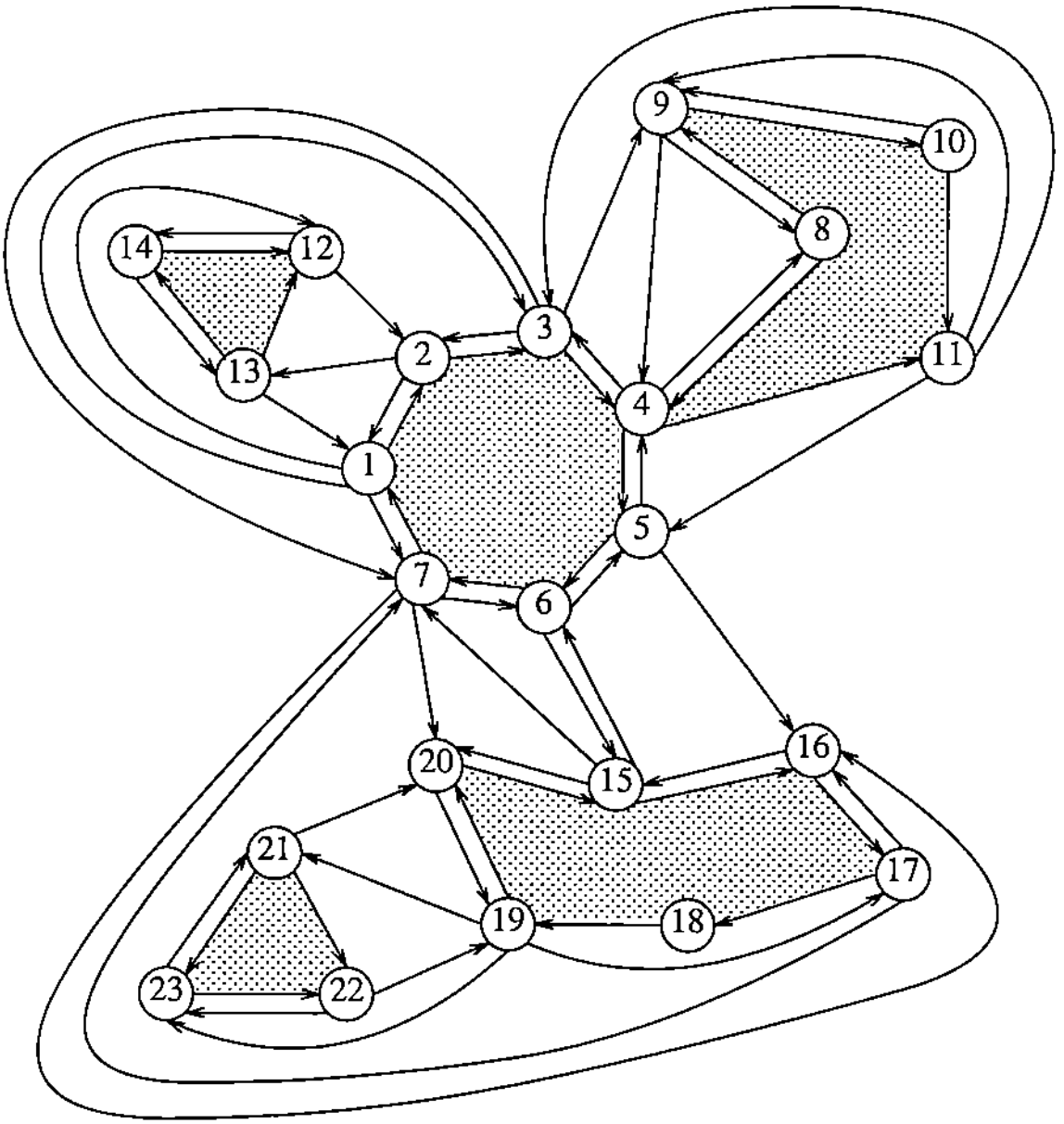



Figure 1. An embedded graph with a face-on-vertex covering of five faces shown as shaded.

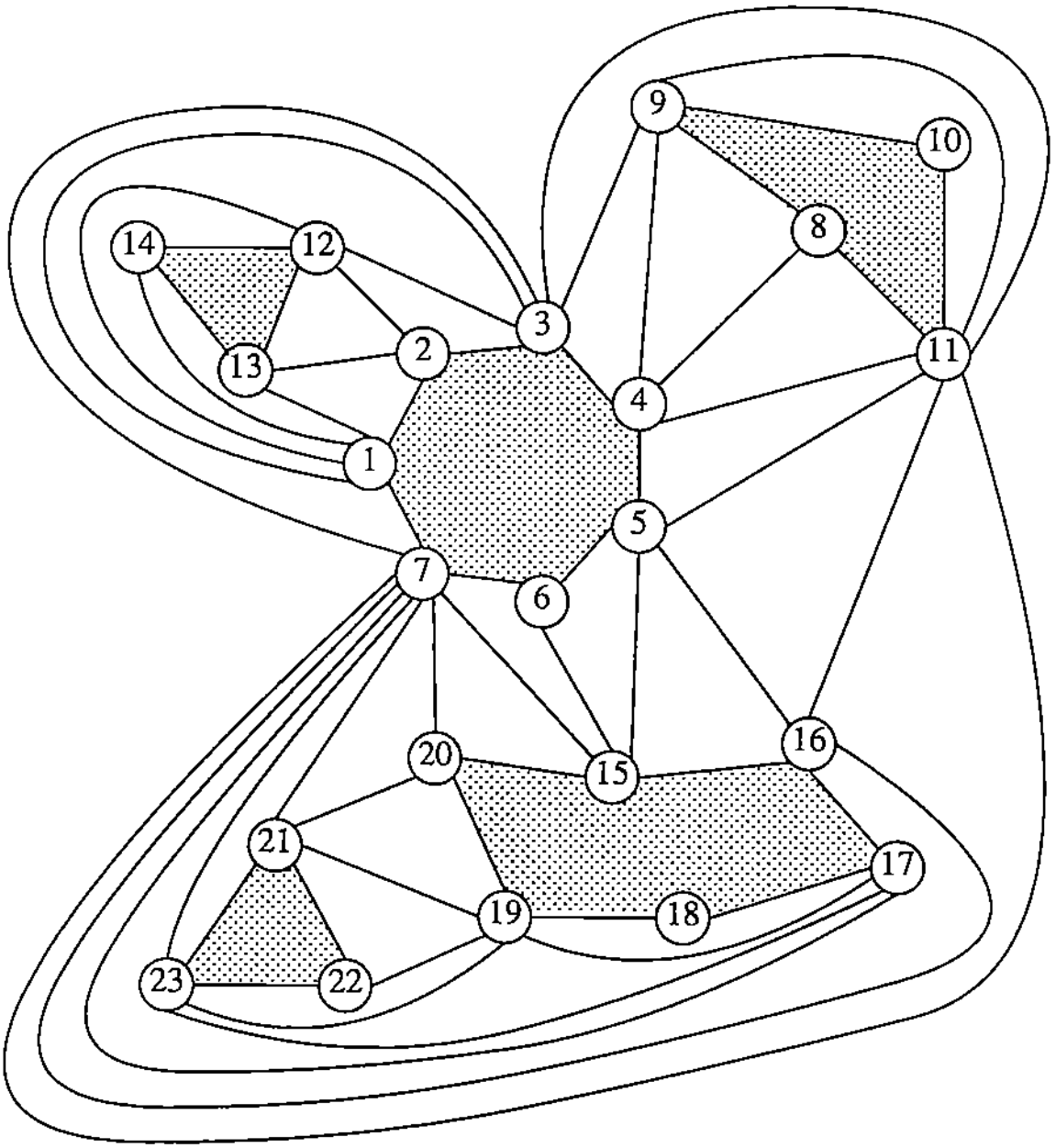



Figure 2. An undirected embedded graph generated from the graph in Figure 1, with faces not in the face-on-vertex covering triangulated.

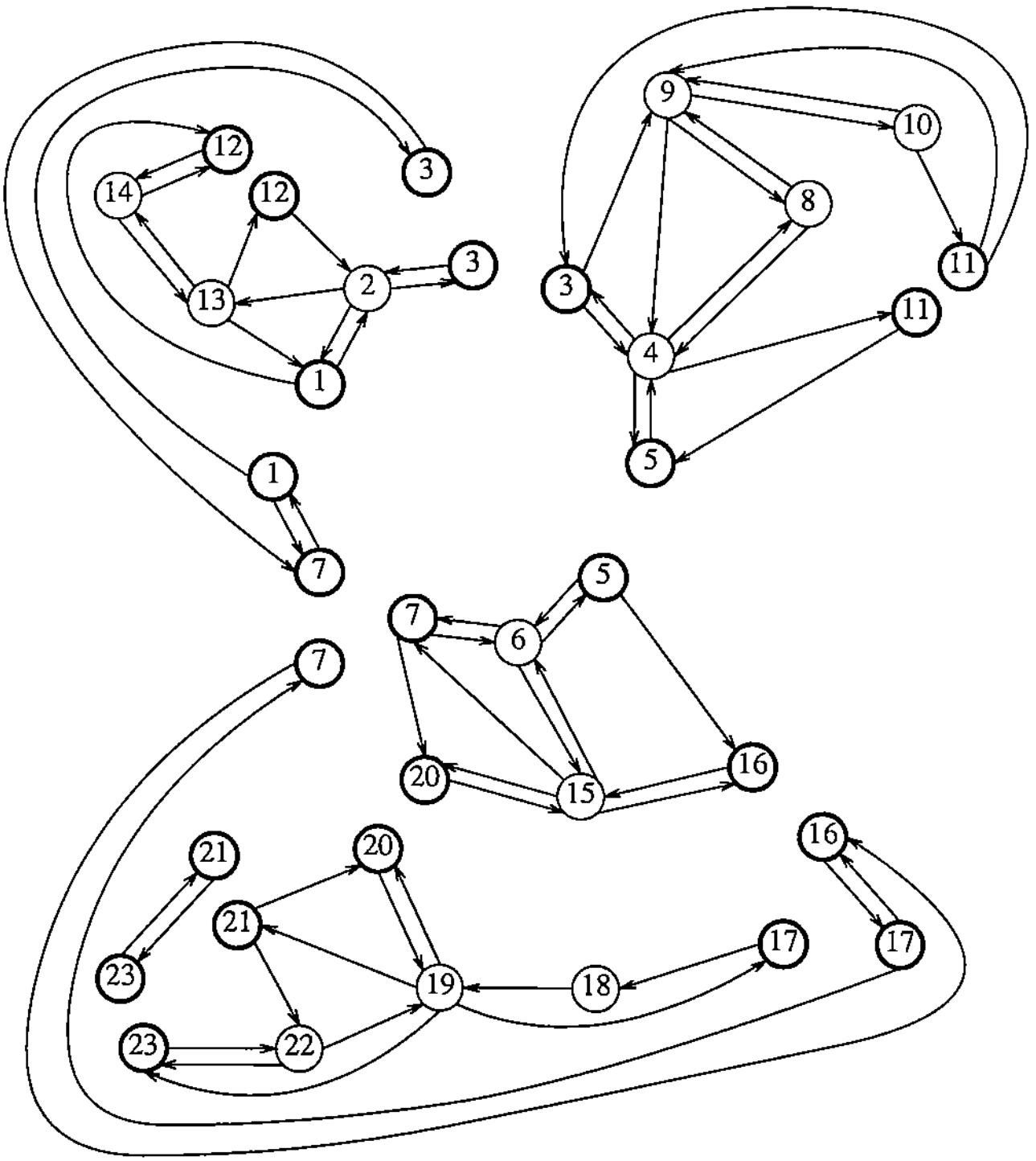



Figure 3. A hammock decomposition for the embedded graph in Figure 1, with the attachment vertices emboldened.

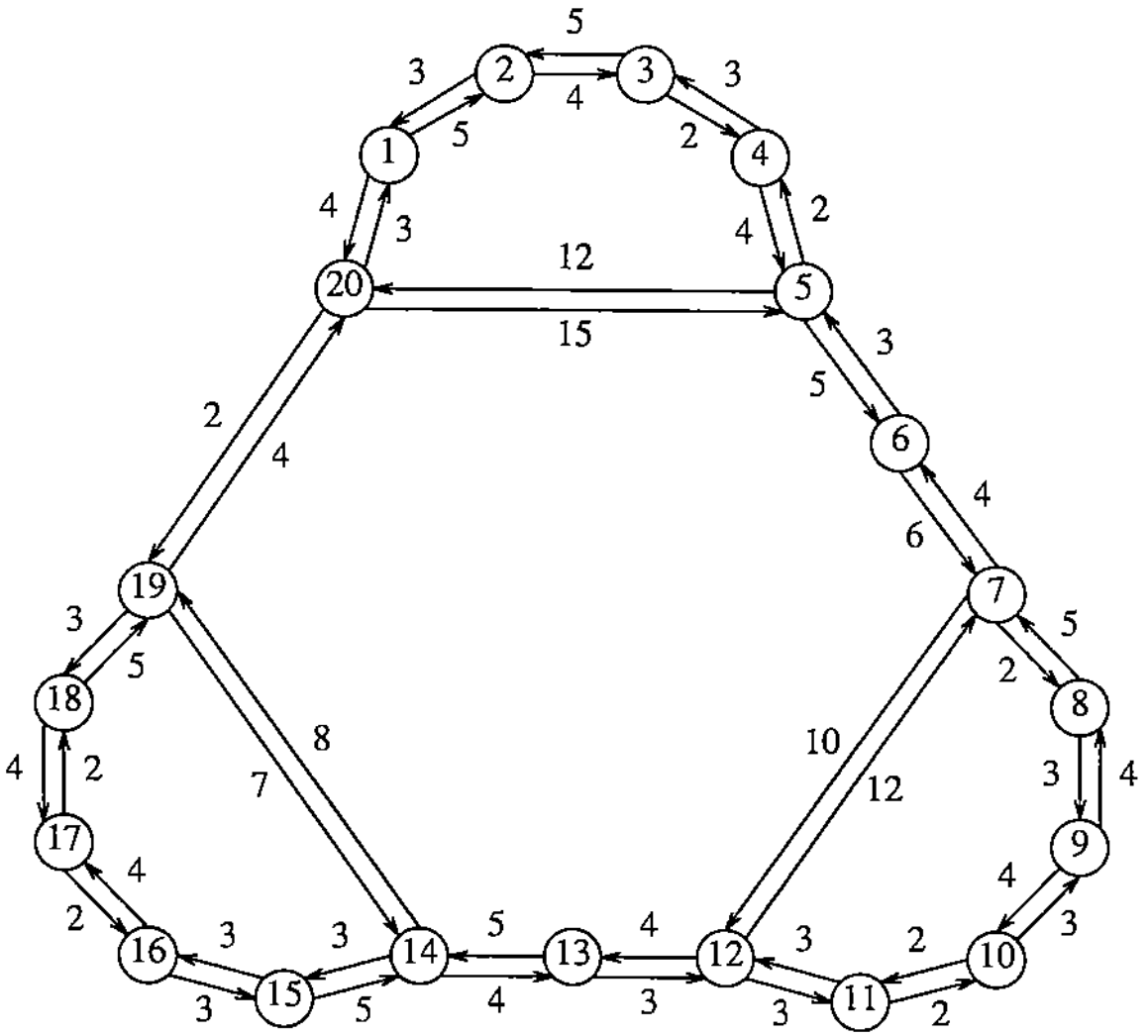
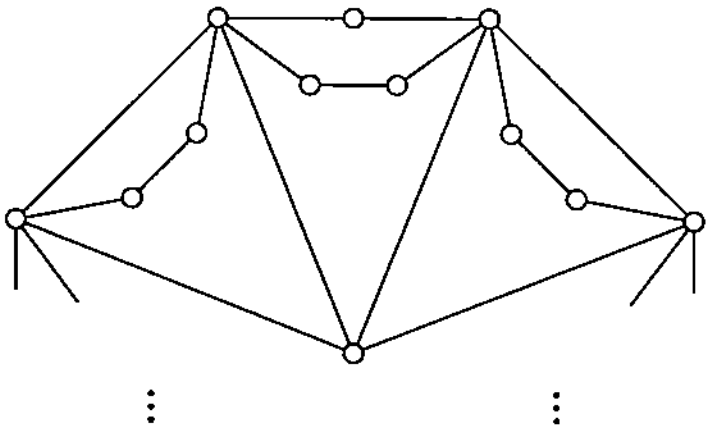
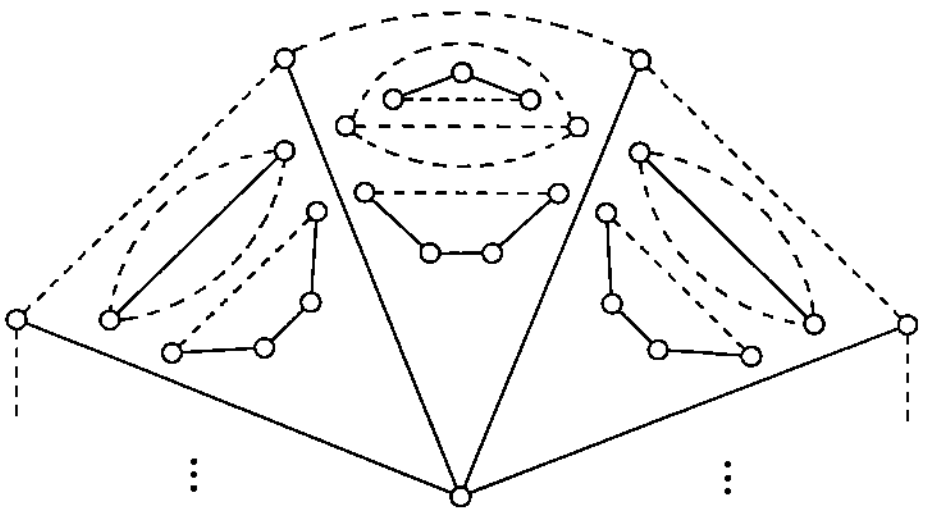



Figure 4. A directed outerplanar graph.



(a)



(b)

Figure 5. An example for determining a good embedding:
(a) an embedded planar graph, and
(b) its decomposition into triconnected components.

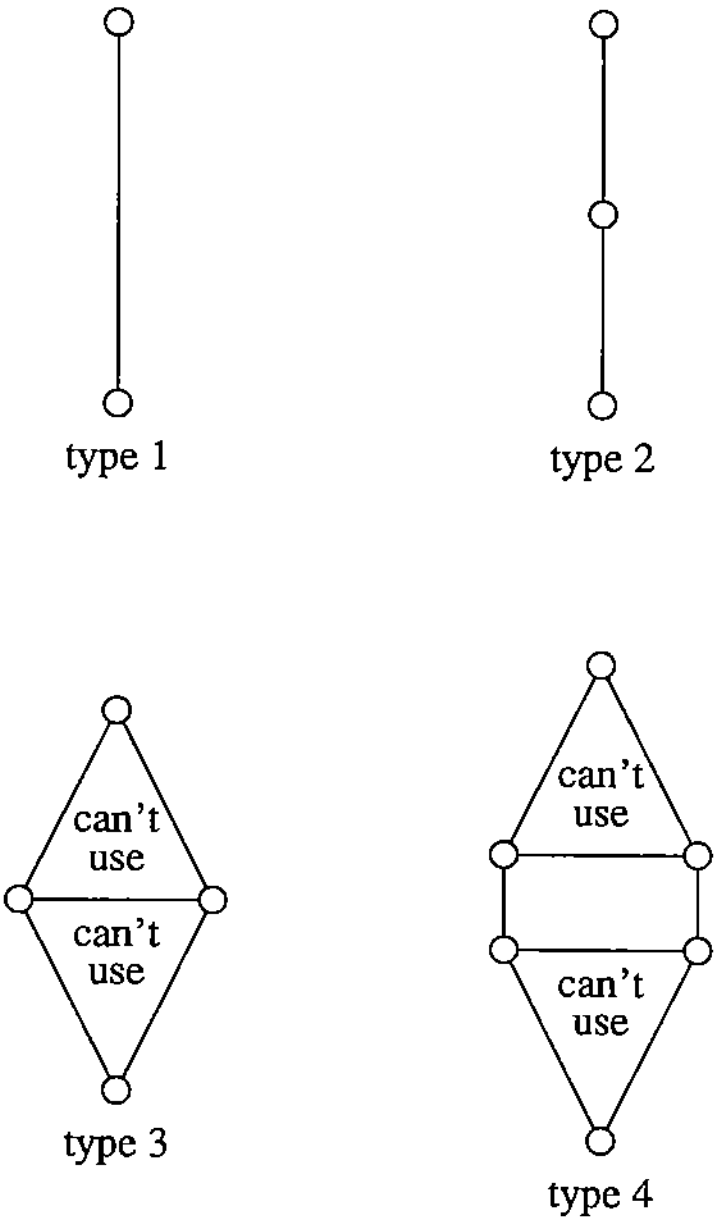
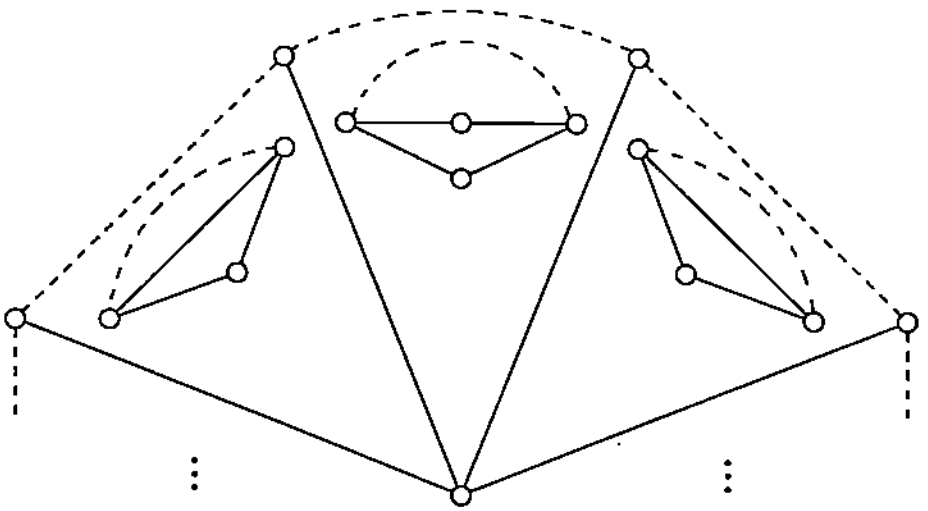
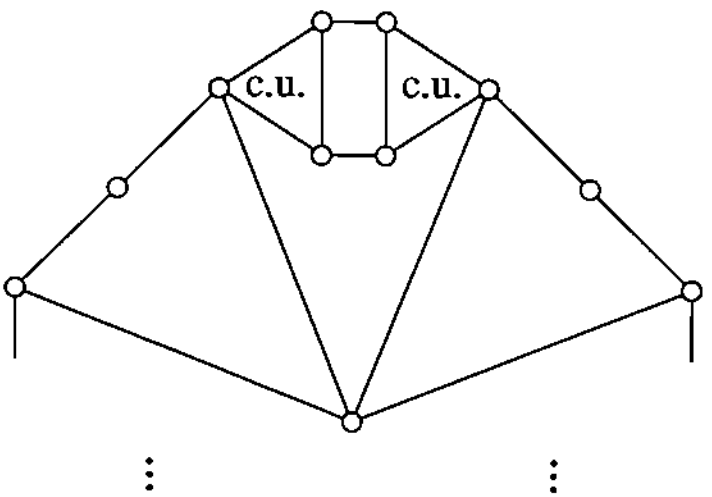



Figure 6. The gadgets that can substitute for a component.

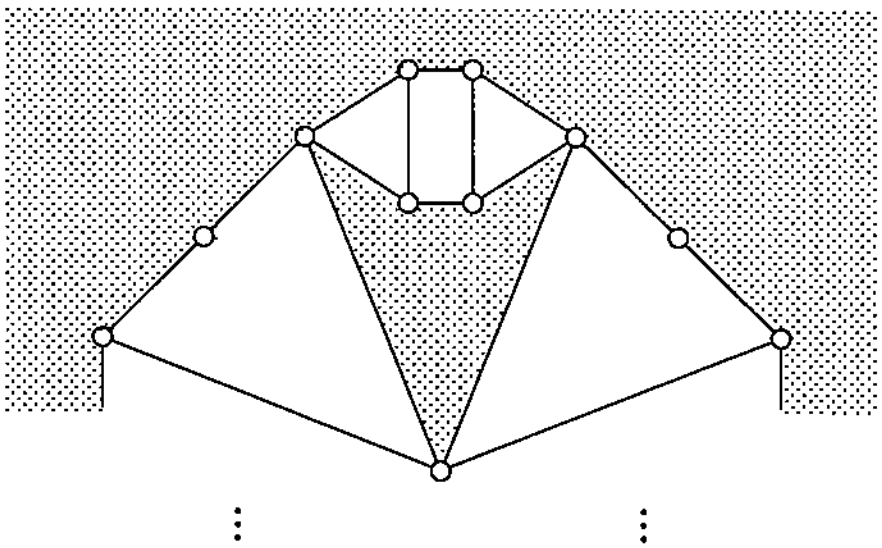


(a)

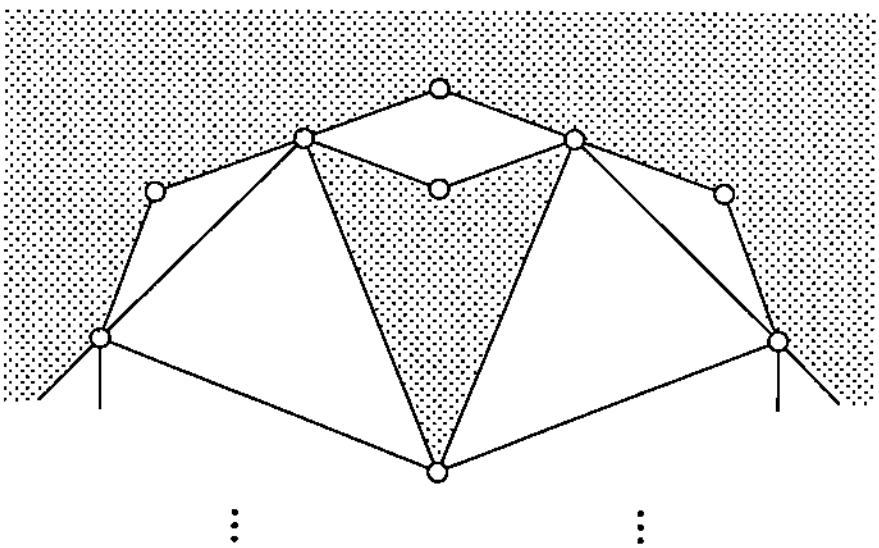



(b)

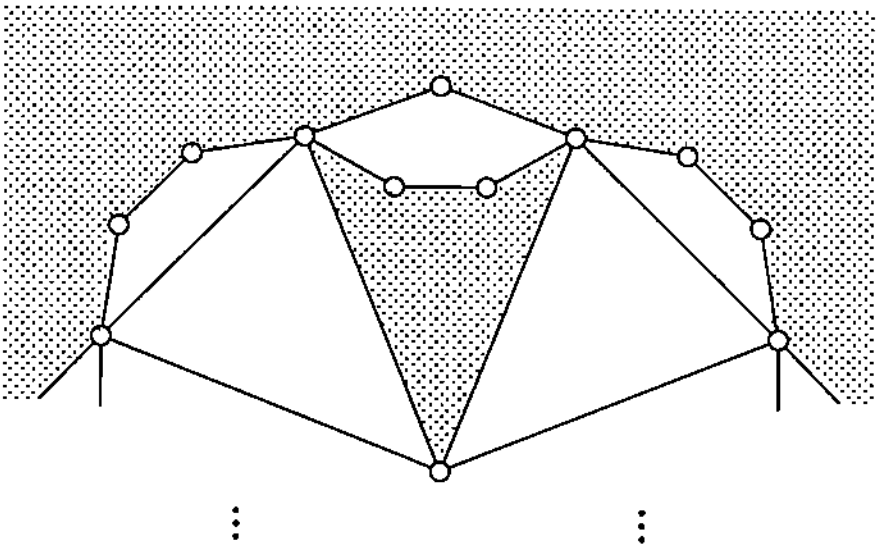
Figure 7. Sets of extended components generated by:
(a) replacing certain components in Fig. 5b , and
(b) replacing all components except the largest one.



(a)



(b)



(c)

Figure 8. Expanding the embedding and face covering:
(a) a face covering for the extended component in Fig. 7b,
(b) a face covering when each gadget in Fig. 7b is expanded, and
(c) a good embedding and face covering for the graph in Fig. 6a.