Hammock-on-Ears Decomposition: A Technique for the Efficient Parallel Solution of Shortest Paths and Other Problems *

Dimitris Kavvadias
Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
and
Department of Mathematics, Patras University, 26500 Patras, Greece
Email: kavadias@cti.gr

Grammati E. Pantziou
Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
and
Department of Mathematics and Computer Science,
Dartmouth College, Hanover NH 03755, USA
Email: pantziou@cs.dartmouth.edu

Paul G. Spirakis
Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
and
Department of Computer Science & Engineering,
Patras University, 26500 Patras, Greece
Email: spirakis@cti.gr

Christos D. Zaroliagis
Max-Planck Institut für Informatik,
Im Stadtwald, 66123 Saarbrücken, Germany
Phone: +49 (681) 302-5356, Fax: +49 (681) 302-5401
Email: zaro@mpi-sb.mpg.de

September 27, 1994

*This work was partially supported by the EEC ESPRIT Basic Reasearch Action No. 7141 (ALCOM II) and by the Ministry of Education of Greece. The work of the second author was also partially supported by the NSF postdoctoral fellowship No. CDA-9211155.

Abstract

We show how to decompose efficiently in parallel *any* graph into a number, $\tilde{\gamma}$, of outerplanar subgraphs (called *hammocks*) satisfying certain separator properties. Our work combines and extends the sequential hammock decomposition technique introduced by G. Frederickson and the parallel ear decomposition technique, thus we call it the *hammock-on-ears decomposition*. We mention that hammock-on-ears decomposition also draws from techniques in computational geometry and that an embedding of the graph does not need to be provided with the input. We achieve this decomposition in $O(\log n \log\log n)$ time using $O(n+m)$ CREW PRAM processors, for an n-vertex, m-edge graph or digraph. The hammock-on-ears decomposition implies a general framework for solving graph problems efficiently. Its value is demonstrated by a variety of applications on a significant class of (di)graphs, namely that of *sparse (di)graphs*. This class consists of all (di)graphs which have a $\tilde{\gamma}$ between 1 and $\Theta(n)$, and includes planar graphs and graphs with genus $o(n)$. We improve previous bounds for certain instances of shortest paths and related problems, in this class of graphs. These problems include all pairs shortest paths, all pairs reachability, and detection of a negative cycle.

1 Introduction

The efficient parallel solution of many problems often requires the invention and use of original, novel approaches radically different from those used to solve the same problems sequentially. Notorious examples are list ranking, connected components, depth-first search in planar graphs, etc (see [3, 6, 24]).

In some cases, the novel parallel solution paradigm stems from a non-trivial parallelization of a specific sequential method (e.g. merge-sort for EREW PRAM optimal sorting [5]). In such cases, this may also lead to more efficient *sequential* algorithms for the same problems. This second paradigm is demonstrated in our paper. Specifically, we provide an efficient parallel algorithm for decomposing *any* (di)graph into a set of outerplanar subgraphs (called *hammocks*). We call this technique the *hammock-on-ears decomposition*. As the name indicates, our technique is based on the sequential hammock decomposition method of Frederickson [13, 14] and on the well-known ear decomposition technique [30], and non-trivially extends our previous work for planar digraphs [36] to any digraph. We demonstrate its applicability by using it to improve the parallel (and sequential) bounds for a variety of problems in a significant class of (di)graphs, namely that of *sparse (di)graphs*. This class consists of all n-vertex (di)graphs which can be decomposed into a number of hammocks, $\tilde{\gamma}$, ranging from 1 up to $\Theta(n)$ (or alternatively have an $O(n)$ number of edges). This class includes planar graphs and graphs whose genus is bounded by any function $\gamma(n) = o(n)$.

The hammock-on-ears decomposition (like the sequential hammock decomposition) decomposes any (di)graph G into a number of outerplanar subgraphs (the *hammocks*) that satisfy certain separator conditions. The decomposition has the following properties: (i) each hammock has at most *four* vertices in common with any other hammock (and therefore with the rest of the graph), called the *attachment vertices*; (ii) each edge of the graph belongs to exactly one hammock; and (iii) the number of hammocks produced is order of the minimum possible among all decompositions and is bounded by a function involving certain topological measures of G (genus, or crosscap number). We achieve this decomposition in two major phases. In the first phase, whose outcome are outerplanar portions of the graph (called outerplanar outgrowths), we transform an initial arbitrary ear decomposition into a new one whose ears include with certainty the outerplanar outgrowths. Then by employing techniques from parallel computational geometry, we identify in each ear the outerplanar outgrowths if they exist. In the second phase, we identify the hammocks by using the output of phase one and by performing some local degree tests.

This decomposition allows us to partially reduce the solution of a given problem Π on G, to the solution of Π in an outerplanar graph. The *general scheme* for solving problems using the hammock-on-ears decomposition technique consists of the following major steps:

1. Find a hammock-on-ears decomposition of the input (di)graph G, into a set of $\tilde{\gamma}$ hammocks.

2. Solve the given problem Π in each hammock separately.

3. Generate a compressed version of G of size $O(\tilde{\gamma})$, and solve Π in this compressed (di)graph using an alternative method.

4. Combine the information computed in steps 2 and 3 above, in order to get the solution of Π for the initial (di)graph G.

The above scheme was used in a specific way in many sequential [11, 14, 13] and parallel applications for planar digraphs [11, 36], but was not employed as a general framework for solving problems. We apply this scheme to the following problems and achieve considerable improvements in the case of sparse digraphs: all pairs shortest paths (apsp), detection of a negative cycle and all pairs reachability (apr).

1.1 Overview of previous work and motivation

The sequential hammock decomposition technique was developed by Frederickson in [13, 14]. In the sequel, let $G = (V(G), E(G))$ denote any (di)graph, and let $n = |V(G)|$ and $m = |E(G)|$. The number $\tilde{\gamma}$ of hammocks produced was shown [13] to be $\tilde{\gamma} = \Theta(\gamma(G')) = \Theta(\bar{\gamma}(G'))$ where $\gamma(G')$ and $\bar{\gamma}(G')$ are the genus and crosscap number [19] of a graph G', respectively. Here G' is G with a new vertex v added and arcs from v to every vertex of G. Moreover, $\gamma(G') \leq \gamma(G) + q$ where G is supposed to be embedded on an orientable surface of genus $\gamma(G)$ such that all vertices are covered by at most q of the faces[1]. Therefore, $\tilde{\gamma}(G)$ can range from 1 up to $\Theta(m)$ depending on the topology of the graph. Both decompositions are achieved in $O(n+m)$ time.

Hammock decomposition seems to be an important topological decomposition of a graph which proved useful in solving efficiently apsp problems [13, 14, 36], along with the idea of storing shortest path information into compact routing tables (succinct encoding) [15, 45]. (In this representation each edge $\langle v, w\rangle$ is associated with at most $\tilde{\gamma}$ disjoint subintervals of $[1, n]$ such that $\langle v, w\rangle$ is the first edge in a shortest path from v to every vertex in these subintervals.) Compact routing tables are very useful in space-efficient methods for message routing in distributed networks [15, 45]. The main benefit from this idea is the beating of the $\Omega(n^2)$ sequential lower bound for apsp (if the output is required to be in the form of n shortest path trees or a distance matrix) when $\tilde{\gamma}$ is small. Frederickson showed how to produce such an encoding in $O(n\tilde{\gamma})$ time for planar digraphs [14] and in $O(n\tilde{\gamma}+\tilde{\gamma}^2\log\tilde{\gamma})$ time for sparse digraphs [13]. Thus, efficient parallelization of this decomposition (along with other techniques) may lead to a number of processors much less than $M_s(n)$ (i.e. the number required by the best known parallel algorithm that uses the matrix powering method to solve apsp in time $O(\log^2 n)$ on a CREW PRAM), and hence beat the so-called *transitive closure bottleneck* [25]. Such a "beating", for planar digraphs, was first achieved in [36]. (The best value, up to now, for $M_s(n)$ is $O(n^3(\log\log n)^{1/3}/(\log n)^{7/6})$ [21].) If the digraph is provided with an $O(n^{\mu})$-separator decomposition[2], $0<\mu<1$, it has recently been shown by Cohen [4] that one can find apsp in $O(\log^3 n)$ time and $O((n^2+n^{2\mu+1})/\log^3 n)$ processors on an EREW PRAM. For the case of planar digraphs, where an $O(n^{1/2})$-separator decomposition can be computed in $O(\log^5 n)$ time using $O(n^{1+\epsilon})$ CREW PRAM processors for every $0<\epsilon<1$ [16], the results in [4] imply an $O(\log^5 n)$-time, $O(n^2/\log^5 n)$-processor CREW PRAM algorithm for the apsp problem. (According to the best of our knowledge, it is not known yet how to compute in general $O(n^{\mu})$-separators in NC.)

In the case where a digraph G has negative edge-costs, it is well-known [42] that there is a shortest path from a vertex v to a vertex u in G iff no path from v to u contains a negative cycle (i.e. a simple cycle for which the sum of its edge-costs is negative). Hence, detecting a negative cycle is an essential problem for finding shortest paths in G. The best previous sequential algorithms for solving the negative cycle problem in general digraphs run in $O(nm)$ time [8, 41, 42]. In the case where the digraph has an $O(n^{\mu})$-separator

[1] We say that a face f *covers* a set $S \subseteq V(G)$ of vertices, if every vertex in S is incident to f. Here q is the minimum number of faces that together cover all vertices of G.

[2] A separator of a digraph G is a subset of vertices whose removal disconnects G into components such that each component has size at most a constant fraction of $|V(G)|$. An $f(n)$-separator decomposition is a recursive decomposition of G using separators, where subgraphs of size n have separators of size $O(f(n))$.

decomposition, the best previous algorithm is due to Mehlhorn & Schmidt [33] and runs in $O(n^{3\mu} + n^{1+\mu} \log n)$ time. In parallel computation, the main tool used was the matrix powering method (using e.g. the algorithm described in [27]), which means that we need $O(\log^2 n)$ time and $M_s(n)$ processors on a CREW PRAM. In the case where the digraph is provided with an $O(n^{\mu})$-separator decomposition, the best previous algorithm was given in [4] and runs in the same resource bounds with the apsp algorithm presented there.

The apr problem is also a fundamental problem in digraphs with both theoretical and practical applications [8, 24, 39]. In parallel computation, even in the case of sparse digraphs, the best previous algorithm uses the matrix powering method and hence needs $O(\log^2 n)$ time by employing $M_r(n)$ CREW PRAM processors [23]. (The best value for $M_r(n)$ is $O(n^{2.376})$ [7]. The best sequential algorithm runs in $O(\min\{M_r(n), nm\})$ time [8].) Also, if we are provided with an $O(n^{\mu})$-separator decomposition, Cohen in [4] gives an algorithm for the apr problem which runs in the same resource bounds with the one for the apsp problem presented in that paper.

It is evident by the above discussion that all previous parallel algorithms, especially in the case of non-planar sparse digraphs, perform considerably more work compared with the best known sequential ones. Moreover, in all cases the lower bound of $\Omega(n^2)$ – for the work – seems difficult to beat, even in the case where the digraph has nice topological properties.

We should also note here that all the above problems, besides their own significance, are frequently used as subroutines in many other applications. For example, finding shortest path information in digraphs has applications to network optimization problems (e.g. matching, maximum flow, etc.) [22, 34, 37], as well as to other areas like incremental computation for data flow analysis and interactive systems design [38, 44]. The negative cycle problem has applications to two dimensional package element [29], to min-cost flows [37], to the problem of minimal cost-to-time ratio [27] and to checking constraints in VLSI layout [28]. Applications of the reachability problem can be found in [24, 39]. Therefore, efficient solutions of the apsp, apr and negative cycle problems can also lead to efficient solutions for other problems too.

1.2 Our results

Given a digraph G we can generate a hammock-on-ears decomposition of G consisting of a minimum number of $\tilde{\gamma}$ hammocks in $O(\log n \log\log n)$ time using $O(n+m)$ CREW PRAM processors. We note here that an embedding of G on some topological surface does not need to be provided with the input. An implementation of our hammock-on-ears decomposition algorithm on a CRCW PRAM, runs in $O(\log n)$ time using $O(n+m)$ processors. A sequential implementation runs in $O(n+m)$ time and matches the running time of [13].

We next apply the hammock-on-ears decomposition (along with the general scheme it implies) to the problems discussed earlier and beat the transitive closure bottleneck in the case of sparse digraphs. More precisely we have achieved the following on a CREW PRAM:

(1) We give an algorithm for computing apsp information (into compact routing tables) in any sparse digraph G (with real-valued edge costs, but no negative cycles) in $O(\log^2 n)$ parallel time, by employing $O(n\tilde{\gamma} + M_s(\tilde{\gamma}))$ processors and using $O(n\tilde{\gamma})$ space. In the case of planar digraphs we can achieve further improvements; namely $O(\log^2 n + \log^5 \tilde{\gamma})$ time and $O(n\tilde{\gamma})$ processors, thus being away of optimality by a polylogarithmic factor only. Note that if we use an alternative encoding for apsp information (compact routing tables for hammocks and global tables for apsp information among the hammocks) that needs $O(n + \tilde{\gamma}^2)$ space (and sequential time to be set up) [11, 13, 14], we can achieve further improvements on the processor bounds. Namely, $O(n + M_s(\tilde{\gamma}))$ processors for the general case and if the graph is planar, the processor bound can be further reduced to $O(n + \tilde{\gamma}^2/\log^5 \tilde{\gamma})$.

(2) We overcome the negative cycle assumption mentioned above, by presenting an algorithm to test for the existence of a negative cycle in any sparse digraph G (with real-valued edge costs). Our algorithm runs in $O(\log^2 n)$ parallel time by employing $O(n + M_s(\tilde{\gamma}))$ processors and uses $O(n + \tilde{\gamma}^2)$ space. A sequential implementation of our algorithm runs in $O(n + \min\{\tilde{\gamma}^{3\mu} + \tilde{\gamma}^{1+\mu} \log \tilde{\gamma}, \tilde{\gamma}^2\})$ time, where $0 < \mu < 1$ and G has a separator of size $O(n^\mu)$. The algorithm is based on a novel optimal solution of the same problem when the input digraph is outerplanar (thus tackling step 2 of the general scheme), in $O(\log n \log^* n)$ parallel time by employing $O(n/\log n \log^* n)$ processors. In the case of planar digraphs, we can achieve further improvements; namely $O(\log^2 n + \log^5 \tilde{\gamma})$ time and $O(n + \tilde{\gamma}^2/\log^5 \tilde{\gamma})$ processors. (In this case our sequential implementation runs in $O(n + \tilde{\gamma}^{1.5} \log \tilde{\gamma})$ time.)

(3) We give an algorithm for providing a succinct encoding of apr information (into compact routing tables) in any sparse digraph G, in $O(\log^2 n)$ time with $O(n\tilde{\gamma} + M_r(\tilde{\gamma}))$ processors using $O(n\tilde{\gamma})$ space. In the case of planar digraphs the algorithm runs in the same bounds with those given above for the apsp problem. If we use an alternative encoding for apr information (compact routing tables for hammocks and global tables for apr information among the hammocks) that needs $O(n+\tilde{\gamma}^2)$ space (and sequential time to be set up), we can achieve further improvements on the processor bounds. Namely, $O(n + M_r(\tilde{\gamma}))$ processors for the general case and if the graph is planar, the processor bound can be further reduced to $O(n + \tilde{\gamma}^2/\log^5 \tilde{\gamma})$.

We note that: (i) The hammock-on-ears decomposition fully extends the topological characteristics of the input digraph and can be advantageously used in the design of algorithms which are parameterized in terms of these characteristics. The better the topological characteristics are (i.e. the smaller the $\tilde{\gamma}$), the more efficient the algorithms for the above applications become. (ii) Our parallel algorithms for solving the apsp, apr and negative cycle problems on any digraph with $\tilde{\gamma} = o(n)$, beat the $M_s(n)$ or $M_r(n)$ lower bound on the number of processors for the same problems in this class of graphs. (iii) Our algorithms for all the above problems explicitly construct the graph decomposition. If additionally a separator decomposition is provided with the input, our general scheme guarantees that the algorithms presented here will still provide better results for all of these problems. Note that the bounds in this case are similar to the ones given for planar digraphs and are discussed in Section 5 of this paper. (iv) We are able to test for the existence of a negative cycle in any sparse digraph. In addition, the sequential version of our negative cycle algorithm is clearly an improvement over the algorithms of [33, 41, 42] (for the class of sparse digraphs) and moreover, overcomes the assumption of nonexistence of a negative cycle for the apsp problem in the sequential results of Frederickson [13, 14].

The paper is organized as follows. In Section 2 we give some preliminaries and define the hammocks. Our parallel algorithm for generating the hammock-on-ears decomposition of a graph is presented in Section 3, while its applications to the problems discussed earlier are presented in Section 4. Finally, in Section 5 we conclude and give some further results.

2 Preliminaries

Let G be a digraph with real-valued edge costs. A graph is called *outerplanar* if it can be embedded in the plane so that all the vertices are on one face. Note that $\tilde{\gamma} = q$ if G is planar (since $\gamma(G) = 0$), and $\tilde{\gamma} = 1$ if G is outerplanar (since $q = 1$ in this case). A tree is called *convergent (divergent)* if the edges of the tree point from a node to its parent (children). If there is a (directed) simple path in G from a vertex v to a vertex w, we say that w *is reachable* by v. The *length* of a simple path P is the sum of the costs of all edges in P and the *distance* between two vertices v and w is the minimum length of a path between v and w. Such a path of minimum length is called *shortest path*. A simple cycle C in G is a simple path starting and ending at the same vertex v. If the length of C is smaller than zero, then C is called *negative*. The *all pairs shortest paths problem* asks for finding shortest path information between every pair of vertices s and t in G (provided that there are no negative cycles in G). The *negative cycle problem* asks for detecting if there exists a simple cycle in G of negative length. If it exists, then output such a cycle. The *all pairs reachability problem* asks for finding reachability information for every pair of vertices s and t in G (i.e. if there is a directed path from s to t).

The notion of *compact routing tables* appeared in [15] and is based on ideas of [45]. Let the vertices of G be assigned names from 1 up to n in a manner to be discussed. For each arc $\langle v, w\rangle$, 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 arc on this path being $\langle v,w\rangle$. (In the event of ties, apply a tie-breaking rule so that for each v, u, $v \neq u$, u is in just one set $S(v,w)$). Let each $S(v,w)$ be described as a union of a minimum number of subintervals of $[1,n]$. We allow a subinterval to wrap around i.e. a set $\{i, i+1, ..., n, 1, 2, ..., j\}$ (where $i > j+1$) will be denoted by $[i,j]$. The set $S(v,w)$ will be called the *compact label* of $\langle v,w\rangle$. If G is outerplanar, then $S(v,w)$ is a single interval [15]. Otherwise, $S(v,w)$ can consist of more than one subinterval. A compact routing table will then have an entry for each of the subintervals contained in a compact label at v. Having this information a convergent or divergent shortest path tree can be computed in $O(n)$ sequential time [14], or in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors [36].

An *ear decomposition* $D = \{P_0, P_1, ..., P_{r-1}\}$ of an undirected graph $G = (V, E)$ is a partition of E into an ordered collection of edge-disjoint simple paths $P_0, ..., P_{r-1}$ such that P_0 is an edge, $P_0 \cup P_1$ is a simple cycle and each endpoint of P_i, $i > 1$, is contained in some P_j, $j < i$, and none of the internal vertices of P_i are contained in any P_j, $j < i$. The paths in D are called *ears*. An ear is *open* if it is noncyclic and is *closed* otherwise. A *trivial ear* is an ear containing only one edge. D is an *open ear decomposition* if all of its ears are open.

We define hammocks following [13]. Let G be a digraph whose weakly undirected counterpart is biconnected (else, we work in each biconnected component separately). Let v be a vertex not in G. Let $G' = (V(G'), E(G'))$, where $V(G') = V(G) \cup \{v\}$, $v \notin V(G)$, and $E(G') = E(G) \cup \{\langle v,w\rangle : w \in V(G)\}$. Let $\hat{G}'$ be an embedding of G' in a surface such that if both arcs $\langle v,w\rangle$ and $\langle w,v\rangle$ belong to $\hat{G}'$, then they together bound a face. The hammocks of G will be defined with respect to $\hat{G}'$. First undirect $\hat{G}'$. Next triangulate each face that is bounded by more than three edges in such a way that no additional edges incident on v are introduced. Finally delete v and its adjacent edges, yielding embedding $I(G)$. In $I(G)$ one large face is always created (the *basic face*) containing all the vertices. The remaining faces are all triangles. The resulting $I(G)$ is called a *basic face embedded graph*. Faces are grouped together to yield certain outerplanar graphs called *hammocks* by using two operations: *absorption* and *sequencing*. Absorption can be done by initially marking each edge that borders the basic face. Let f_1, f_2 be two nonbasic faces sharing an edge. Suppose f_1 contains two marked edges. Then absorb f_1 into f_2. (This is equivalent to first contracting one edge that f_1 shares with the basic face. The first face becomes a face bounded by two parallel edges, one of which also belongs to f_2. Then delete this edge, merging f_1 and f_2.) Repeat the absorption until it can no longer be applied.

After the end of absorptions, we group remaining faces by sequencing. Identify maximal sequences of faces such that each face in the sequence has a marked edge and each pair of consecutive faces share an edge in common. Each sequence 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 graph is called a *(major) hammock*. The first and last vertices on each face of the hammock are called *attachment vertices*. Note that there are four attachment vertices per hammock that constitute the only way of "communication" between the hammock and the rest of the graph. Any edge not included in a major hammock is taken by itself to induce a *(minor) hammock*. A *hammock decomposition* of $I(G)$ is the set of all major and minor hammocks. It is not hard to see that the total size of a compact routing table T is $O(n\tilde{\gamma}(G))$ by constructing T through a decomposition of G into a number of $\tilde{\gamma}(G)$ hammocks (see also [15]). Let $\tilde{\gamma}(G)$ be the minimum number of hammocks into which G can be decomposed. Since the direct approach for decomposing G would involve finding an embedding of minimum genus (shown to be NP-complete in [43]), Frederickson used an alternative approach, the so called *partial hammock decomposition*. (A partial hammock is a subgraph of a hammock in some basic face embedding $I(G)$.) This decomposition decomposes G into $O(\tilde{\gamma}(G))$ partial hammocks using two operations: pseudo-absorption and pseudo-sequencing. As their names indicate these operations are analogous to absorption and sequencing (defined above) and in general produce only partial hammocks. The interesting feature of these operations is that they are applied to G without an embedding to work with.

3 The Algorithm for the Hammock-on-Ears Decomposition

Let G_u be the undirected version of a digraph G. We assume that G_u is biconnected, if not, then we work on each biconnected component separately.

Let v_1, v_2 be a separation pair of G_u that separate V_1 from V_2. Let J_1 be the subgraph of G_u induced on $V_1 \cup \{v_1, v_2\}$ and let J be J_1 with the edge $\{v_1, v_2\}$ added. Let G_1 be the graph resulting from contracting all edges with both endpoints in V_1. If J is outerplanar and J_1 is the maximal graph such that J is outerplanar then we call J_1 an *outerplanar outgrowth* of G_u and (G_1, J) an *outerplanar trim* of G_u. The following Lemma has been proved in [13].

Lemma 3.1 *([13]) Let (G_1, J) be an outerplanar trim of a biconnected graph G_u. Then $\tilde{\gamma}(G_1) = \tilde{\gamma}(G_u)$ and a basic face embedded graph for G_1 of minimum hammock number can be extended to a basic face embedded graph for G_u of minimum hammock number.*

The above Lemma actually says that we may remove all outerplanar outgrowths of the graph G_u, find a minimum decomposition of the remaining graph in outerplanar subgraphs (hammocks) and then reinsert the removed outerplanar outgrowths of G_u. The resulting decomposition still consists of outerplanar subgraphs and moreover, the number of hammocks of the decomposition is minimum. This procedure is called *pseudo absorption*

[13]. The first phase of our parallel decomposition algorithm is to efficiently parallelize the pseudo absorption procedure. Because of the sequential nature of this procedure, we had to employ different techniques specifically suited for parallel computation such as the ear decomposition search.

After all outerplanar outgrowths have been identified, they are removed from the graph leaving an edge connecting the separation points of the outgrowth, labeled with sufficient information to rebuild the outgrowth after the decomposition. A second procedure, called *pseudo sequencing* [13] is then applied, in order to identify sufficiently long sequences of faces from each hammock. The second phase of our algorithm is to parallelize the pseudo sequencing procedure.

3.1 Pseudo absorption

The first step in our *pseudo absorption* algorithm, is to create an open ear decomposition of the graph. The key observation is that the first (lower numbered) ear that involves an outerplanar outgrowth enters the outgrowth from one of the separation points and exits from the other. Moreover, all ears of the outerplanar outgrowth whose endpoints are vertices of this first ear, have endpoints that are *consecutive* vertices of this ear, otherwise the outerplanarity assumption would be violated. The same holds for any ear with greater number that is included in the outgrowth: It must "touch" a lower numbered ear in consecutive vertices. For the same reason there are no ears with endpoints in two different ears. The shape of the outerplanar outgrowth is therefore as shown in Fig. 3.1 where v_1, u_1 and v_2, u_2 are pairs of consecutive vertices of the ear P_i.

Based on this observation, we start from an open ear decomposition and try to identify ears that have the above property: Their endpoints are consecutive vertices of another ear. Any such maximal set of ears is a possible outerplanar outgrowth. It is useful to view this set of ears as a new ear, and transform the ear decomposition to a new one where there are no ears having both their endpoints to be consecutive nodes of another ear. Note also (in Fig. 3.1) the possible existence of ears (both trivial and non-trivial) that connect nonconsecutive vertices, which may destroy the outerplanarity and which are treated separately (e.g. the ear with endpoints v_3, u_3 in Fig. 3.1) in the next step. This step consists of retransforming the ear decomposition by dividing each ear P_i into a number of ears such that each new ear is a maximal candidate of being an outerplanar outgrowth. The transformation of the ear decomposition, is done in stage 1 of algorithm *Find_Outgrowths* below.

ALGORITHM Find_Outgrowths. Stage 1:
BEGIN

1.1. Let $D = \{P_0, P_1, ..., P_{r-1}\}$ be an open ear decomposition of G_u.

1.2. Construct an auxiliary graph A as follows: Create a vertex for each ear P_i. For each P_i, if both its endpoints belong to the same ear P_j, $j < i$, and are consecutive vertices of P_j, and moreover there is no other non-trivial ear $P_{i'}$ having these two vertices as endpoints, then let (P_i, P_j) be an edge of A.

1.3. Find the connected components of A. Each connected component is a tree (there is no cycle because each endpoint of an ear P_i belongs to a smaller numbered ear). Note that the root of such a tree is an ear that either has its endpoints on different ears or on the same ear P_k but the endpoints are not consecutive vertices of P_k.

1.4. In each connected component, join the ears of the component into a single ear by converting for each ear the edge between its endpoints into a new trivial ear, and also rearranging the pointers of the vertices that point to the next vertex in the obvious way. The number of the new ear is equal to the number of the ear which is the root of the tree (connected component). Call the new ear decomposition D_1.

1.5 Now call *internal* vertices of an ear P_i, all its vertices except for its endpoints and *internal* ears with respect to P_i all ears (both trivial and non-trivial) whose both endpoints are vertices of P_i. Similarly, ears that have only one endpoint to be a vertex of P_i are called *external*. We call *dividing vertices of stage 1* the endpoints of:

(i) P_i.

(ii) All external non-trivial ears.

(iii) All internal non-trivial ears.

Divide each ear P_i into a set of new ears each having as endpoints two consecutive (on the ear P_i) dividing vertices of stage 1 (see Fig. 3.2). Renumber the vertices of the new ears so as to be consecutively numbered. Call the new ear decomposition D_2.

END.

Stage 1 of the algorithm *Find_Outgrowths* is based on the following Lemma:

Lemma 3.2 *Each non-trivial ear produced from stage 1 of the algorithm Find_Outgrowths is a maximal candidate for being an outerplanar outgrowth together with a possible trivial ear (edge) that has the same endpoints. That is, no outerplanar outgrowth may span the whole or a portion of an ear P_i and a subgraph of the rest of the graph.*

Proof: Let J_1 be an outerplanar outgrowth with separation vertices v_1 and v_2. By definition J, that is, J_1 with the edge $\{v_1, v_2\}$ added, must be outerplanar. After step 1.5 of stage 1 each non-trivial ear P_i of the ear decomposition D_2, has no external and no internal non-trivial ears attached to it and moreover both its endpoints are not internal vertices of another ear. This implies that each endpoint of P_i is also an endpoint of at least two other ears which begin at that endpoint. Consequently, if at least one endpoint of P_i say vertex v, is included in J_1 but it is not a separation point, these ears must also be included in J_1 otherwise additional links of J_1 exist and v_1, v_2 are not separation points. Also observe that all but at most one of the ears that begin at v (except for P_i) must be simple edges otherwise J could not be outerplanar. Now if P_i includes one of the separation points of J then these edges must be connected to vertices of P_i. Since v was found to be dividing at least one of those edges was part of another ear before step 1.5, as internal trivial ears do not define dividing vertices. This implies not only that this particular edge is connected to the other endpoint of P_i which necessarily must be a separation vertex, but also that P_i has endpoints that where consecutive vertices of another ear. This possibility was ruled out at step 1.4 and thus we come to a contradiction. If P_i is totally included in J_1, then similar discussion leads again to the fact that the two dividing vertices that define P_i, where consecutive vertices of another ear before step 1.5, a contradiction. It follows that the only remaining possibility is to have v_1 and v_2 coincide with the endpoints of P_i. Observe that this construction does not rule out the possibility of an ear to have external trivial ears attached to it and thus further splitting may be necessary. Also there could be a case where there exist other ears with the same endpoints v_1, v_2. A non-trivial ear with endpoints v_1, v_2 could not be included in J_1 since then J could not be outerplanar. A trivial ear could be included if the subgraph induced on P_i is outerplanar. ■

In the beginning of the second stage we have a new ear decomposition where each non-trivial ear P_i is a maximal candidate for being an outerplanar outgrowth. An ear P_i could have however external trivial ears or internal edges attached to it that may destroy the outerplanarity and hence only subsets (if any) of the subgraph attached to P_i could be outerplanar outgrowths. This second stage resolves this problem.

Now call *dividing vertices of stage 2* the endpoints of:

(i) P_i and of all external trivial ears.

(ii) All internal trivial ears (edges) that intersect with other internal or external trivial ears. Two internal ears with endpoints v_1, v_2 and u_1, u_2 intersect, if exactly one endpoint of the second ear (e.g. exactly one of u_1 and u_2) lies between the endpoints of the first one on the ear P_i. In the case where one is external consider it as being connected to one endpoint of P_i. Equivalently, if we consider the numbers assigned to the nodes of P_i at the end of stage 1, two ears intersect if the corresponding intervals intersect and no interval is a subset of the other.

(iii) All internal trivial ears that do not intersect with other ears and whose endpoints are separated by at least one dividing vertex of cases (i) and (ii).

ALGORITHM Find_Outgrowths. Stage 2:
BEGIN

2.1 For all ears P_i (produced by stage 1) in parallel, locate the dividing vertices of types (i) to (iii) defined above.

2.2 Subgraphs of P_i that are separated by two consecutive dividing vertices are outerplanar outgrowths (see Fig. 3.3). Delete each such subgraph and substitute it by a single edge that connects these dividing vertices. In order to be able to easily reconstruct the outgrowths, we label the edge by the numbers of the vertices of the corresponding outgrowth.
END.

Stage 2 of the algorithm is based on the following Lemma:

Lemma 3.3 *The subgraphs induced on each of the portions into which the dividing vertices separate an ear P_i are outerplanar outgrowths (provided of course, that are not simple edges). The separation vertices of an outgrowth are the two dividing vertices that define the portion.*

Proof: The subgraph induced on a portion defined by two consecutive dividing vertices contains (by definition) the corresponding vertices and edges of the ear P_i, and all other edges having those vertices as endpoints, no two of which intersect with each other, in an embedding where P_i is arranged as a straight line segment. Moreover, there are no edges connecting a vertex in a given portion with the rest of the graph. If there were such an edge, then it would fall into cases (i) or (iii) and its endpoints would also be dividing vertices. A graph defined in this way is clearly outerplanar and is separated from the rest of the graph by the two dividing vertices. It is maximal, since the addition of any vertices must necessarily include at least one of the dividing vertices that define J. This vertex now becomes internal to J (i.e. it is no longer a separation vertex of J). Now if this vertex was found to be dividing because it falls into case (i) then a portion of at least one more ear must be included in the outerplanar outgrowth which by Lemma 3.2 is impossible. If it is of type (ii) then it is the endpoint of an intersecting edge. This edge must also be included in J, otherwise a link with the rest of the graph is introduced. Now this edge intersects with another which has an endpoint between the endpoints of the first. This endpoint must be also included in J. This follows from the fact that in order to separate this endpoint from J at least two more separation points are needed. For the same reason the intersecting edge must also be included in J, thus destroying the outerplanarity. If the vertex was found to be dividing because it falls in case (iii) then it is the one endpoint of an edge which has a dividing vertex of types (i) or (ii) between its endpoints. As before this dividing vertex must be included in J and consequently at least one pair of intersecting edges, again a contradiction. ■

Locating dividing vertices of type (i) is straightforward. Also, if we determine dividing vertices of type (ii), it is easy to determine dividing vertices of type (iii). In order however to locate the dividing vertices of type (ii) we need to locate the intersecting edges. Considering the numbering of the vertices of P_i, we conclude that this problem is equivalent to the following one: Given a set of (open) intervals on $[1, n]$ determine all intervals that intersect with at least another interval.

In our problem we need to determine all ears that are simple edges and intersect with other edges. The endpoints of the edges are the boundaries of the intervals, considering the ear P_i to be the line of numbers. This is actually a geometric problem and we manage to solve it in parallel time $O(\log m)$ using $O(m)$ processors where m is the number of intervals involved. We note here that the literature of parallel geometry is rich in related problems that study intersections of line segments in the plane (see e.g. [2, 18]). However the versions of the problems studied there, usually report *all* intersections and carry therefore the burden of the size of the output either on the time or the processor bounds. In our case we merely want to report intervals that have some intersection.

We briefly now discuss how to solve this problem using a data structure mentioned in [31] called "priority search tree". A priority search tree of the points $(x_i, y_i), i = 1, \ldots, m$ is a binary tree whose nodes keep the points sorted from left to right according to their x coordinate and which is a heap wrt the y coordinate (i.e. a node always keeps a value that is greater than the value kept in its children). Now let $I = (x_1, x_2)$ be an interval chosen from a set of m intervals S of $[1, n]$. We say that I has a *right intersection* in S (left intersection) if there is an interval $I' = [x'_1, x'_2] \in S$ such that $x_1 < x'_1 < x_2$ and $x'_2 > x_2$ (corresp. $x_1 < x'_2 < x_2$ and $x'_1 < x_1$). The following Lemma holds:

Lemma 3.4 *Let R be a priority search tree constructed on S (where the intervals are seen as points). Then given an interval $I \subseteq [1, n]$ we can test whether I has a right intersection in S in sequential time $O(\log m)$. Symmetrically, we treat left intersections by a priority search tree that has reversed the heap property. Moreover, constructing R in parallel can be accomplished in $O(\log m)$ time using $O(m)$ CREW PRAM processors.*

Proof: Follows from a well-known reduction to the problem of locating all points on a plane that lay in a half-infinite band with sides parallel to the x, y-axes, see [32]. In the same reference, the construction of the priority search tree basically requires a sorting of the points according to their x-coordinate and a knock-out tournament in order to establish the heap property of the y-coordinate. It is not difficult to see that both steps can be parallelized within the resource bounds stated in the Lemma. ■

Therefore we have:

Lemma 3.5 *The problem of interval intersections can be solved in $O(\log m)$ parallel time by employing $O(m)$ CREW PRAM processors.*

Proof: By Lemma 3.4 the tree R on S can be constructed in $O(\log m)$ time using $O(m)$ CREW PRAM processors. Given this tree, we can solve interval intersections in the same resource bounds by letting each processor to query one interval in parallel. ∎

Theorem 3.1 *Algorithm Find_Outgrowths correctly identifies all outerplanar outgrowths of an n-vertex, m-edge biconnected graph $G = (V, E)$ in $O(\log n)$ time using $O(n + m)$ CRCW PRAM processors or in $O(\log n \log\log n)$ time using $O(n + m)$ CREW PRAM processors.*

Proof: In the first stage of the algorithm, subsets of ears that possibly form outerplanar outgrowths are identified. Each one of these subsets is contracted into one new ear. Next the ear is divided into portions by invoking stage 2. Lemma 3.3 guarantees that the portions are outerplanar outgrowths. Stage 1 of the algorithm employs an open ear decomposition procedure, a connected components procedure and also at steps 1.4 and 1.5 a list ranking procedure. Stage 2 can be done in $O(1)$ time except for the identification of the dividing vertices, which by Lemma 3.5 is achievable in $O(\log n)$ time using $O(n+m)$ CREW PRAM processors. Therefore, the time and processor bounds are dominated by the bounds of the connected components algorithm [3, 40] (which also dominate the bounds for finding an open ear decomposition [30]), and which are those stated in the Theorem. ∎

At this point a remark about the algorithm *Find_Outgrowths* is necessary. Note that the algorithm correctly identifies an outerplanar outgrowth if the first (lower numbered) ear that involves the outgrowth enters from one separation point and exits from the other. The first ear however (the ear P_0) may have one or both endpoints on an outgrowth. In this case one or two outerplanar outgrowths may not be identified correctly. This can be corrected if we first run the algorithm from an arbitrary ear decomposition, identify all but at most two outerplanar outgrowths and then rerun the algorithm from a different ear decomposition with the endpoints of the new P_0 placed in an already discovered outgrowth. This can be done since most of the parallel ear decomposition algorithms begin from a spanning tree which also defines the first ear (see [30] for details).

3.2 Pseudo sequencing

This subsection shows how to efficiently parallelize the *pseudo sequencing* procedure. The parallel algorithm presented is based on ideas developed in [13] where also the main theory behind the procedure can be found. The goal is to identify sequences of faces that cover all but a constant part of a hammock. The hammocks thus produced are called *partial hammocks.*

Let G_{u1} be the undirected graph resulting from the pseudo absorption procedure. Each edge in G_{u1} has a label indicating the vertices that have been contracted, between its endpoints. Note that all vertices in G_{u1} have degree greater than 2.

Consider a hammock H in a basic face embedded graph $I(G)$ corresponding to a subgraph H' that results from performing the edge contractions in generating G_{u1}. Assume that H' is biconnected. The non biconnected case is handled in a preprocessing step to be discussed later. Now observe that since every node of G_{u1} has degree at least three and H' is outerplanar it follows that all faces of H' are bounded by either three or four edges (there can be a situation where this is not true, involving the attachment vertices of the hammock which however only represent a constant part). This observation suggests that a hammock is actually a sequence of triangles and rectangles sharing an edge in a way that outerplanarity is preserved. Consequently, two neighboring faces of a hammock form a graph that is isomorphic to one of the three graphs P_1, P_2 or P_3 of Fig. 3.4. We will next try to identify *interior* edges of a hammock (these are the edges that separate two faces in the same hammock, i.e. the edges (v_1, v_2) of the previous graphs).

The following Lemma is a combination of Lemmata 6.2, 6.3 and 6.4 that were proved in [13].

Lemma 3.6 *Any sequence of* 7 *faces in a hammock (i.e. combination of triangles and rectangles) falls into at least one of* 10 *possible cases. At least one of the graphs* P_1*,* P_2 *or* P_3 *of Fig. 3.4 appears in any of the* 10 *cases having at least two of its vertices with degree less that* 7*. This graph can be identified in any of the cases by applying one of the following degree tests:*
(1) P_1 *with* $deg(v_2) = deg(v_3) = deg(v_4) = 3$.
(2) P_1 *with* $deg(v_2) = deg(v_3) = 3$ *and* $deg(v_4) = 4$.
(3) P_1 *with* $deg(v_2) = 3$ *and* $deg(v_1) = 5$.
(4) P_1 *with* $deg(v_2) = 3$ *and* $deg(v_1) = 6$.
(5) P_1 *with* $deg(v_2) = deg(v_1) = 4$.
(6) P_1 *with* $deg(v_2) = 4$ *and* $deg(v_3) = deg(v_4) = 3$.
(7) P_2 *with* $deg(v_1) = 4$ *and* $deg(v_2) = deg(v_4) = 3$.
(8) P_2 *with* $deg(v_1) = 4$ *and* $deg(v_2) = deg(v_5) = 3$.
(9) P_3 *with* $deg(v_1) = deg(v_2) = deg(v_4) = deg(v_6) = 3$.
(10) P_3 *with* $deg(v_1) = deg(v_2) = deg(v_3) = deg(v_6) = 3$.

The importance of the above Lemma lies in the fact that it shows first that any part of a hammock consisting of seven faces has a constant number of different forms, and second that each one of these forms is identifiable by a single processor in constant time after running the ten tests sequentially. So a portion of a hammock is identifiable in constant time by a single processor. After identifying it, we next delete edges that are interior to the hammock. If a vertex of degree two results from the deletion then it is *contracted*, i.e we remove it and join its two neighbors by an edge, having a label with that vertex in order to be able to rebuild the hammock. It is important to mention here that after the deletion of an edge and the subsequent contraction of vertices (if any) the hammock is still a sequence of triangles and rectangles and, hence, the same tests can be applied until all the hammock has been shrunk. The following parallel algorithm uses these observations to identify edges that are interior to hammocks.

ALGORITHM Find_Sequences
BEGIN

Repeatedly apply steps 1-3 for $3 \times \lceil \log n \rceil$ steps.

1. Use one processor for each vertex and check in turn the 10 cases mentioned above in order to identify a portion of a hammock. Note that this task involves the scanning of an adjacency list of a vertex of length at most six.

2. Identify an internal edge and remove it by doing a contraction operation. Note here that several processors may identify the same edge. However their number will be constant and hence breaking ties can be done in constant time.

3. Delete all vertices having degree 2 by applying the contraction operation.

END

Lemma 3.7 *Let G be a directed graph such that for each hammock H in a basic face embedding, the corresponding H' in G_{u1} is biconnected. Then, algorithm Find_Sequences generates a labeling of the edges that gives a partial hammock decomposition of $O(\tilde{\gamma}(G))$ hammocks. The algorithm takes $O(\log n)$ time and uses $O(n+m)$ processors on a CREW PRAM.*

Proof: In each iteration of the *Find_Sequences* procedure at least one edge for every seven faces in every hammock is contracted. Since each contraction joins two neighboring faces in one, we conclude that $\frac{1}{7}$th of a hammock is contracted in every step. Thus, after at most $3 \times \lceil \log n \rceil$ iterations all hammocks will be contracted. Consider now the graph resulting from the *Find_Sequences* procedure. Each label of an edge in this graph, represents a sequence of vertices around the basic face in some embedding. The vertices are listed around one of the two faces of a partial hammock. Each hammock H' has now at most 6 faces, i.e. constant number of edges. Thus the number of partial hammocks generated with respect to a given hammock H is a constant. This means that we have a partial hammock decomposition of $O(\tilde{\gamma}(G))$ hammocks. ∎

All the approach above was based on the assumption that the hammocks were biconnected. If however a hammock is not biconnected the above procedure may delete an external edge of the hammock. To handle the general case, we preprocess G_{u1} to add in dummy vertices and edges so that each hammock becomes biconnected. This addition is done by identifying the two possible subgraphs that a hammock must contain in order to be non biconnected (see [13]). This operation is also done by using local degree tests and is therefore not difficult to parallelize. We call this parallel algorithm *Preprocess*. We therefore have:

Theorem 3.2 *Given an n-vertex, m-edge digraph* G*, the decomposition of* G *into* $O(\tilde{\gamma}(G))$ *partial hammocks can be done in* $O(\log n \log\log n)$ *time using* $O(n+m)$ *CREW PRAM processors, or in* $O(\log n)$ *time using* $O(n+m)$ *CRCW PRAM processors.*

Proof: The algorithm to find a partial hammock decomposition of a digraph consists of the following steps: First, undirect G and find its biconnected components (using the algorithms in [3, 40]). Second, find the outerplanar outgrowths in the undirected version of G (algorithm *Find_Outgrowths*). Third, make the appropriate preprocessing such that the hammocks of the graph are biconnected (algorithm *Preprocess*). Fourth, find the sequences that make up the partial hammocks (algorithm *Find_Sequences*). By Theorem 3.1, Lemma 3.7 and the discussion preceding the Theorem, it is clear that the dominating steps in terms of time are the first two ones, which in turn are based on the algorithms of [3, 40] (for finding connected components in parallel) as it was also discussed in the proof of Theorem 3.1. ∎

4 Applications

In this section we give a high level description of our algorithms for the problems mentioned in the introduction. Also, recall the general scheme for problem solving using the hammock-on-ears decomposition technique. In the sequel, let $G = (V, E)$ be a sparse digraph with real-valued edge costs. We will use $\tilde{\gamma}$ instead of $\tilde{\gamma}(G)$ for notational simplicity.

4.1 Computing all pairs shortest paths information

The algorithm for computing apsp information in G is as follows:

ALGORITHM Sparse_Apsp
BEGIN

1. Find a hammock-on-ears decomposition of G into $O(\tilde{\gamma})$ hammocks.

2. Rename the vertices of G, in a way such that all vertices belonging in a hammock are consecutively numbered around its external face. (This is needed for the encoding of apsp information into compact routing tables.)

3. Find apsp information in each hammock.

4. Find apsp in G between each pair of attachment vertices of the hammocks. This is done as follows. For each hammock H, generate a compressed version $C(H)$. This is done using the algorithm in [36] to find the shortest distance between each pair of attachment vertices of H. The graph $C(H)$ is just a complete digraph on the attachment vertices in H (which are at most four), with the cost of each edge being the shortest distance in H between its endpoints. Then generate a *compressed version* $C(G)$ of G by replacing each hammock H by its compressed version $C(H)$ and deleting all but one least expensive copy of any multiple edge. The compressed graph $C(G)$ will have $O(\tilde{\gamma})$ vertices and edges. Find apsp in $C(G)$ by running another algorithm.

5. For each pair of hammocks determine succinct shortest paths information for each vertex in one hammock to all vertices in the other hammock.

6. For each hammock determine shortest path information between vertices in the same hammock. (This is needed since a shortest path between two vertices in a hammock may leave and reenter the hammock.)

END.

Theorem 4.1 *Given an n-vertex sparse digraph G with real-valued edge costs (but no negative cycles), that can be decomposed into a minimum number of $O(\tilde{\gamma})$ hammocks, we can encode all pairs shortest paths information into compact routing tables in $O(\log^2 n)$ parallel time using $O(n\tilde{\gamma} + M_s(\tilde{\gamma}))$ CREW PRAM processors and $O(n\tilde{\gamma})$ space.*

Proof: Note that step 1 of algorithm *Sparse_Apsp* corresponds to major step 1 of the general scheme, steps 2 and 3 correspond to major step 2, step 4 corresponds to major step 3 and steps 5 and 6 correspond to major step 4. Now, for the resource bounds the following hold in a CREW PRAM. Step 1 needs $O(\log n \log\log n)$ time using $O(n)$ processors by Theorem 3.2. Step 2 can be done in parallel by sorting in a straightforward way in $O(\log n)$ time using $O(n)$ processors. Step 3 can be implemented in $O(\log^2 n)$ time using $O(n)$ processors by [36]. Step 4 needs $O(\log n)$ time with $O(n/\log n)$ processors for the generation of $C(H)$ [36], and $O(\log^2 \tilde{\gamma})$ time using $M_s(\tilde{\gamma})$ processors for apsp in $C(G)$ by running the algorithm of [21]. Step 5 needs for all hammocks $O(\log n)$ time using $O(n\tilde{\gamma})$ processors by [36]. Finally, step 6 (again by [36]) needs $O(\log^2 n)$ time using $O(n)$ processors. The space bound follows by the discussion in Section 2. ∎

Corollary 4.1 *Given an n-vertex planar digraph G with real-valued edge costs (but no negative cycles), that can be decomposed into a minimum number of $O(\tilde{\gamma})$ hammocks, we can* *encode all pairs shortest paths information into compact routing tables, in* $O(\log^2 n + \log^5 \tilde{\gamma})$ *parallel time using* $O(n\tilde{\gamma})$ *CREW PRAM processors and* $O(n\tilde{\gamma})$ *space.*

Proof: Follows by Theorem 4.1, if in step 4 of algorithm *Sparse_Apsp* we run the algorithm of [4] instead of the one in [21] and the algorithm of [36] for generating the graph $C(H)$ which is outerplanar and of $O(1)$ size. (This latter step takes $O(\log n)$ time using $O(n)$ processors.) ■

Corollary 4.2 *Given an* n*-vertex sparse digraph* G *with real-valued edge costs (but no negative cycles), that can be decomposed into a minimum number of* $O(\tilde{\gamma})$ *hammocks, we can compute all pairs shortest paths information using an alternative encoding (compact routing tables for hammocks and global tables for shortest paths among the attachment vertices of the hammocks), in* $O(\log^2 n)$ *parallel time using* $O(n + M_s(\tilde{\gamma}))$ *CREW PRAM processors and* $O(n + \tilde{\gamma}^2)$ *space.*

Proof: Note that using the alternative encoding it is not necessary to run steps 5 and 6 in algorithm *Sparse_Apsp*, as it is described in [11, 14]. A shortest path or distance between two vertices v and w is computed as follows. If both vertices belong to the same hammock H we have first to compute the distance between them inside H and then compare it with the minimum among the distances $d(v, a_i) + d(a_i, a_j) + d(a_j, w)$, where $a_i \neq a_j$ and a_i, $i = 1, 2, 3, 4$, are the four attachment vertices of H. If w belongs to another hammock H', then we have to choose those i and j that minimize the quantity $d(v, a_i) + d(a_i, b_j) + d(b_j, w)$, where b_j, $j = 1, 2, 3, 4$, are the attachment vertices of H'. ■

Corollary 4.3 *Given an* n*-vertex planar digraph* G *with real-valued edge costs (but no negative cycles), that can be decomposed into a minimum number of* $O(\tilde{\gamma})$ *hammocks, we can compute all pairs shortest paths information using an alternative encoding (compact routing tables for hammocks and global tables for shortest paths among the attachment vertices of the hammocks), in* $O(\log^2 n + \log^5 \tilde{\gamma})$ *parallel time using* $O(n + \tilde{\gamma}^2 / \log^5 \tilde{\gamma})$ *CREW PRAM processors and* $O(n + \tilde{\gamma}^2)$ *space.*

4.2 Detecting negative cycles

In this section we give an efficient algorithm for detecting a negative cycle in G (and outputting it if it exists). The algorithm is based on the hammock-on-ears decomposition technique and on the detection of a negative cycle in an outerplanar digraph. We shall first discuss the outerplanar case and then the general one.

4.2.1 An optimal work algorithm for detecting negative cycles in outerplanar digraphs

In this section we will show how to solve optimally the negative cycle problem provided that the input digraph is outerplanar and biconnected. (If it is not biconnected, apply the algorithm to every biconnected component.) This is achieved by using a novel extension of the fundamental tree-contraction technique [1, 25] to the tree of interior faces of the outerplanar digraph.

Let $G_o = (V, E)$ be the input outerplanar digraph and let $\hat{G}_o$ be its undirected version. It is well known that the dual graph of $\hat{G}_o$ is a tree, called the *tree of faces*. (The exterior face is excluded in this construction.) We assume that all vertices in G_o are named consecutively in clockwise order around the exterior face and the tree of faces in $\hat{G}_o$ is binary. At the end of the section we will show how to overcome these assumptions.

Let $s_1(f)$ (resp., $s_2(f)$) be the minimum (resp., maximum) numbered vertex in each interior face (or group of neighboring faces) f. We call these vertices, the *associated vertices* of f and the arc(s) $\langle s_1(f), s_2(f)\rangle$ ($\langle s_2(f), s_1(f)\rangle$) the *associated arc(s)* of f. A pair of vertices v, w is called a *separation pair* in $\hat{G}_o$, if their removal disconnects $\hat{G}_o$. If $\{v, w\}$ is an edge then it is called a *separation edge* or *separator*. A path from a vertex v to a vertex w in a subgraph J of G_o will be denoted by $P(v, w; J)$. A shortest path from v to w in J is denoted by $SP(v, w; J)$. In the sequel, by $\{v, w\}$ we denote the singleton $\{\langle v, w\rangle\}$, or $\{\langle w, v\rangle\}$, or their union depending on the existence of those arcs in G_o. Furthermore, all references to $\{v, w\}$ are considered to be references to all its members.

Consider the following *problem* Π: Let G be an outerplanar digraph and $\{v, w\}$ be a separation edge, separating G into two subgraphs G_1 and G_2. Suppose also that in each G_i, $i = 1, 2$, there is no negative cycle. Find a negative cycle in G (if it exists).

Solution of Π: since there is no negative cycle in each G_i, a possible negative cycle $N(G)$ for G will consist of a path in G_1 joined with a path in G_2. Also, $N(G)$ will have v and w as two of its vertices. In order to find $N(G)$ it suffices to find $SP(v, w; G_1)$ (or $SP(w, v; G_1)$) and $SP(w, v; G_2)$ (or $(SP(v, w; G_2))$. The union of these two paths will give (the possible) $N(G)$. Therefore we have the following.

Proposition 4.1 *Let $G = (V, E)$ be an outerplanar digraph and let $\{v, w\}$ be a separator, separating G into two subgraphs G_1 and G_2. Suppose also that in each G_i, $i = 1, 2$, there is no negative cycle and that the two shortest paths between v and w are known. Then G can be tested for a negative cycle in $O(1)$ time.*

The main idea of the algorithm is the following. We assume that at a certain point, a set of neighboring faces has been tested for a negative cycle. In the case of a positive answer, the negative cycle is output and the algorithm stops. Otherwise, this set of faces is joined to a neighboring set of faces that has also been tested, in order to form a new set. Detection of a negative cycle in this union is done according to the rules stated in Proposition 4.1. Thus, the algorithm proceeds in a bottom-up fashion.

We say that an interior face f has been *evaluated* in the tree of faces T, iff in the subgraph induced by its descendant nodes in T we have tested if there is a negative cycle and in the case of a negative answer, we have computed shortest path information between certain pairs of vertices in f. The main goal is to evaluate the root face of T.

The parallel tree-contraction algorithm [1, 25] evaluates the root of a tree T processing a logarithmic number of binary trees $T_0, T_1, ..., T_k$, $k = O(\log|T|)$, $T_0 = T$ and T_k contains only one node. Also, $|T_i| \leq \varepsilon|T_{i-1}|$, $0 < \varepsilon < 1$. The tree T_i is obtained by T_{i-1} by applying a local operation, called $SHUNT$ [25], to a subset of the leaves of T_{i-1}. The $SHUNT$ operation consists in turn by two other operations, called *prune* and *bypass* [1].

Prune operation: Let l be a leaf in tree T_{i-1}. Let also v and l' be its parent and sibling respectively. Then by "pruning l" we denote the deletion of l from T_{i-1}.

Bypass operation: Let l' be the unique child of a non-root node v in T_{i-1}. Then by "bypassing l'" we denote the joining of l' and v into a new node v'.

To complete the description of the tree-contraction algorithm we have to show how the information concerning the negative cycle is maintained. Let $D(x, y; G)$ denote the set $\{SP(x, y; G), SP(y, x; G)\}$ and let $D((x_1, x_2), (y_1, y_2); G)$ denote the set $\{D(x_i, y_i; G)|i, j \in \{1, 2\}\}$.

Lemma 4.1 *Let G be an outerplanar digraph and let $\{v, w\}$ be a separator of G, separating it into two subgraphs G_1 and G_2. Let also $\{a, b\}$ and $\{c, d\}$ be edges belonging to G_1 and G_2 respectively, and such that $b = a-1$ and $d = c-1$ (assuming consecutive clockwise naming of vertices in G). Suppose that the following shortest paths in G_1, G_2 are known: $D(a, b; G_1)$, $D(v, w; G_1)$, $D(v, w; G_2)$, $D(c, d; G_2)$, $D((a, b), (v, w); G_1)$ and $D((v, w), (c, d); G_2)$. Then in $O(1)$ time we can compute $D(a, b; G)$, $D(c, d; G)$, and $D((a, b), (c, d); G)$.*

Proof: (see Fig. 4.1) Notice that

$$SP(a, b; G) = \min_{length}\{SP(a, b; G_1), SP(a, v; G_1) + SP(v, w; G_2) + SP(w, b; G_1)\}$$

$$SP(c, d; G) = \min_{length}\{SP(c, d; G_2), SP(c, w; G_2) + SP(w, v; G_1) + SP(v, d; G_2)\}$$

$$SP(a, d; G) = \min_{length}\{SP(a, v; G_1) + SP(v, d; G_1), SP(a, w; G_2) + SP(w, d; G_2)\}$$

where "$\min_{length}$" denotes minimum in length and "+" denotes path concatenation. Similar expressions hold for the rest of the requested shortest paths. Note that each one of the above computations involves a constant number of additions and comparisons of the lengths of at most 4 shortest paths, and after that, a constant number of pointer manipulations (to get the shortest path itself). The claimed bound follows now by the fact that the number of the requested shortest paths is also constant. ■

Lemma 4.2 *Let G be an outerplanar digraph and let $\{v, w\}$ be a separator of G, separating it into two subgraphs G_1 and G_2. Let $\{a, b\}$ and $\{c, d\}$ be edges of G_2 such that $b \leq c < d \leq w < v \leq a$ (assuming consecutive clockwise naming of vertices in G). Suppose that the following shortest paths are provided with the input: $D(a, b; G_2)$, $D(v, w; G_1)$, $D(c, d; G_2)$, $D((a, b), (c, d); G_2)$, $D((a, b), (v, w); G_2)$ and $D((c, d), (v, w); G_2)$. Then in $O(1)$ time we can compute $D(a, b; G)$, $D(c, d; G)$ and $D((a, b), (c, d); G)$.*

Proof: (see Fig. 4.2) Notice that

$$\begin{aligned} SP(a,b;G) &= \min_{length}\{SP(a,b;G_2), SP(a,v;G_2) + SP(v,w;G_1) + SP(w,b;G_2)\} \\ SP(c,d;G) &= \min_{length}\{SP(c,d;G_2), SP(c,v;G_2) + SP(v,w;G_1) + SP(w,d;G_2)\} \\ SP(a,c;G) &= \min_{length}\{SP(a,c;G_2), SP(a,v;G_2) + SP(v,w;G_1) + SP(w,c;G_2)\} \end{aligned}$$

Similar expressions hold for the other requested shortest paths. Using a similar argument with the one used in the proof of Lemma 4.1, it is clear that the above computations can be done in $O(1)$ time. ■

Remark: Clearly, the above Lemmata hold in the case where the vertices around the exterior face of G constitute – in clockwise order – a constant number of intervals (instead of one).

The algorithm executes a number of main steps as they are described in e.g. [1] or [25]. This means that some leaves of the tree of faces T perform a *SHUNT* operation (main step) and this is repeated for a logarithmic number of phases. It is worth noting that as the algorithm proceeds and the tree is contracted, each leaf of T corresponds to a set of neighboring faces whose removal does not disconnect G_o. Conversely, an internal node of T corresponds to a set of faces whose removal disconnects G_o. Therefore it remains to explain what information is exchanged and/or updated during a *SHUNT* operation.

We will distinguish between two types of *SHUNT* operations depending on what is the sibling of the tree node performing this operation. In the sequel, all references to a tree node z will be considered also as references to the subgraph of G_o corresponding to z.

TYPE-1: suppose that l_i (see Fig. 4.3) performs a *SHUNT* operation. Suppose also that l_i' is a leaf. Initially we know $D(s_1(l_i), s_2(l_i); l_i)$, $D(s_1(l_i), s_2(l_i); f_i)$, $D(s_1(f_i), s_2(f_i); f_i)$, $D(s_1(l_i'), s_2(l_i'); f_i)$, $D(s_1(l_i'), s_2(l_i'); l_i')$, $D((s_1(f_i), s_2(f_i)), (s_1(l_i), s_2(l_i)); f_i)$, and

$D((s_1(l_i'), s_2(l_i')), (s_1(l_i), s_2(l_i)); f_i)$ (provided that these paths exist, since some of the six associated vertices of l_i, f_i, f_i' may coincide). During the prune operation examine if there exists a negative cycle in $l_i \cup f_i$ (using Proposition 4.1). If it exists then stop, report that a negative cycle was found and output the cycle. Otherwise, compute $D(s_1(l_i'), s_2(l_i'); l_i \cup f_i)$ and $D(s_1(f_i), s_2(f_i); l_i \cup f_i)$ using Lemma 4.2, and continue with the bypass operation resulting into a new node f_i', where $f_i' = l_i \cup f_i \cup l_i'$. Check again if there exists a negative cycle in f_i' and if not, compute $D(s_1(f_i), s_2(f_i); f_i')$ using Lemma 4.1. Note that after this *SHUNT* operation (and in the case where a negative cycle was not found) we have $s_1(f_i') = s_1(f_i)$ and $s_2(f_i') = s_2(f_i)$.

TYPE-2: suppose that l_k performs a *SHUNT* operation, and f_j, the sibling of l_k, is an internal node of T (see Fig. 4.4). Initially we know $D(s_1(l_k), s_2(l_k); l_k)$, $D(s_1(l_k), s_2(l_k); f_k)$, $D(s_1(f_k), s_2(f_k); f_k)$, $D(s_1(f_j), s_2(f_j); f_k)$, $D(s_1(f_j), s_2(f_j); f_j)$, $D(s_1(f_i), s_2(f_i); f_j)$, $D(s_1(l_j), s_2(l_j); f_j)$, $D((s_1(f_k), s_2(f_k)), (s_1(l_k), s_2(l_k)); f_k)$, $D((s_1(f_j), s_2(f_j)), (s_1(l_k), s_2(l_k)); f_k)$, $D((s_1(f_k), s_2(f_k)), (s_1(f_j), s_2(f_j)); f_k)$, $D((s_1(f_j), s_2(f_j)), (s_1(l_j), s_2(l_j)); f_j)$, $D((s_1(f_j), s_2(f_j)), ((s_1(f_i), s_2(f_i)); f_j)$ and $D((s_1(f_i), s_2(f_i)), (s_1(l_j), s_2(l_j)); f_j)$ (provided that these paths exist, since some of the ten associated vertices involved may coincide). During the prune operation examine if there exists a negative cycle in $l_k \cup f_k$ (using Proposition 4.1). If it exists then stop, report that a negative cycle was found and output the cycle. Otherwise, compute $D(s_1(f_j), s_2(f_j); l_k \cup f_k)$ and $D(s_1(f_k), s_2(f_k); l_k \cup f_k)$ using Lemma 4.2. After that continue with the bypass operation. This results into a new node f_{kj}, where $f_{kj} = l_k \cup f_k \cup f_j$. Check again if there exists a negative cycle in f_{kj} and if not, compute $D(s_1(f_k), s_2(f_k); f_{kj})$, $D(s_1(l_j), s_2(l_j); f_{kj})$ and $D((s_1(f_k), s_2(f_k)), (s_1(l_j), s_2(l_j)); f_{kj})$ using one application of Lemma 4.1. Then compute $D(s_1(f_i), s_2(f_i); f_{kj})$ and $D((s_1(f_i), s_2(f_i)), (s_1(f_k), s_2(f_k)); f_{kj})$ using a second application of Lemma 4.1. Furthermore, compute $D((s_1(f_i), s_2(f_i)), (s_1(l_j), s_2(l_j)); f_{kj})$. (Note that this latter set of shortest paths can be easily computed from $D((s_1(f_i), s_2(f_i)), (s_1(l_j), s_2(l_j)); f_j)$, $D((s_1(f_i), s_2(f_i)), ((s_1(f_j), s_2(f_j)); f_j)$ and $D(s_1(f_j), s_2(f_j); l_k \cup f_k)$.) Also, after this *SHUNT* operation (and in the case where a negative cycle was not found) we have $s_1(f_{kj}) = s_1(f_k)$ and $s_2(f_{kj}) = s_2(f_k)$.

This completes the description of the *SHUNT* operations.

Lemma 4.3 *The SHUNT operations are correct and are executed in $O(1)$ time.*

Proof: Consider first the *TYPE*-1 *SHUNT* operation. It is clear by Proposition 4.1 that if there is no negative cycle in any one of l_i, f_i or l_i', then a negative cycle (if exists) would be either in $l_i \cup f_i$ or in $f_i \cup l_i'$ or in f_i'. From the description of this *SHUNT* operation, it is clear that the first and the third case are correctly checked. Note however, that the second case is also checked implicitly in the third one. This is true because a negative cycle in $f_i \cup l_i'$ would involve $D(s_1(l_i'), s_2(l_i'); f_i)$ and $D(s_1(l_i'), s_2(l_i'); l_i')$, according to Proposition 4.1. But notice that the length of any shortest path in $D(s_1(l_i'), s_2(l_i'); l_i \cup f_i)$ is always less than or equal to the length of its corresponding shortest path in $D(s_1(l_i'), s_2(l_i'); f_i)$. Therefore, if a negative cycle exists in any one of $l_i \cup f_i$, $f_i \cup l_i'$ or f_i', then it is correctly detected. In the case that a negative cycle does not exist, observe that f_i' represents a set of faces whose removal does not disconnect G_o. Hence, it is enough to compute the shortest paths between the associated vertices of f_i' (inside f_i'), since by Proposition 4.1 a portion of a negative cycle that passes through f_i' would definitely involve these vertices as well as the shortest path between them.

Consider now the *TYPE*-2 *SHUNT* operation. Assume as before that there are no negative cycles in any one of l_k, f_k and f_j. Using a similar argument as before we can show that a negative cycle (if exists) is correctly detected in $l_k \cup f_k$ or in $f_k \cup f_j$ or in f_{kj}. If such a cycle does not exist, then observe that f_{kj} represents a set of faces whose removal disconnects G_o. Therefore, according to Proposition 4.1, a portion of a negative cycle which pass through f_{kj} would definitely pass through the associated vertices of it and/or through the associated vertices of the two subgraphs corresponding to its two children in the tree, and also would involve the shortest paths among them. Hence, in this case it is sufficient to compute shortest paths among the above mentioned associated vertices (inside f_{kj}).

The resource bound follows clearly by the description of the *SHUNT* operations, Proposition 4.1 and Lemmata 4.1 and 4.2. ∎

Now, we will show how to overcome the assumptions made in the beginning of this section and prepare G_o for the application of the parallel tree-contraction method using the *SHUNT* operations described above. We call this, the *initialization step* of our algorithm. This step includes renaming of vertices in clockwise order around the exterior face, construction of T and binarization, and initial computation of shortest paths in each internal face of G_o. The initialization step consists of the following stages.

1. Rename the vertices of the input biconnected outerplanar digraph $G_o = (V, E)$ such that the new names appear in clockwise order around the exterior face.

2. Triangulate each interior face of $G_o = (V, E)$ (if G_o is not already triangulated) by adding appropriate arcs. Associate with these new arcs a very large cost (e.g. the sum of the absolute values of all edge-costs in G_o) such that they will not be used by any shortest path. Then, construct the tree of faces T (which will be binary since every face is a triangle).

3. In each face f:

(a) Find its associated vertices $s_1(f)$ and $s_2(f)$, and compute the clockwise and counterclockwise distances from $s_1(f)$ to every other vertex in f (i.e. the sum of the costs of the edges in clockwise and counterclockwise order).

(b) compute the following shortest paths: $D(s_1(f), s_2(f); f)$, $D(s_1(f), s_2(f); parent(f))$, where $parent(f)$ is the face corresponding to the parent node of f in T.

(c) if l is the face corresponding to the left child of f and h is the face corresponding to the right child of f in T (see Fig. 4.5), compute the following shortest paths: $D((s_1(f), s_2(f)), (s_1(h), s_2(h)); f)$, $D((s_1(f), s_2(f)), (s_1(l), s_2(l)); f)$ and $D((s_1(h), s_2(h)), (s_1(l), s_2(l)); f)$.

This completes the description of the initialization step and the description of the algorithm. Let us call the algorithm presented in this section *Out_Neg_Cycle*. If G_o is not biconnected, apply *Out_Neg_Cycle* to every biconnected component.

Theorem 4.2 *Given an n-vertex outerplanar digraph $G_o = (V, E)$ with real-valued edge costs, algorithm Out_Neg_Cycle detects and outputs a negative cycle in G_o, if it exists, in $O(\log n \log^* n)$ time using $O(n/\log n \log^* n)$ CREW PRAM processors and $O(n)$ space. A sequential implementation of the algorithm runs in $O(n)$ time.*

Proof: The correctness of the algorithm comes from Lemma 4.3. Now for the resource bounds we have that each main step (*SHUNT* operation) of the algorithm needs $O(1)$ time and $O(|T_i|)$ CREW PRAM processors, where $|T_i|$ is the size of the tree of faces during the i-th phase. This results in a total of $O(\log n)$ time using $O(n/\log n)$ CREW PRAM processors [1]. The bounds of the initialization step are as follows: stage 1 needs $O(\log n \log^* n)$ time and $O(n/\log n \log^* n)$ CREW PRAM processors [36]. Stage 2 needs the same resource bounds with those of stage 1, by [9]. Stage 3(a) can be done in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors by using the algorithm of [6]. Finally stages 3(b) and 3(c) can be easily done having the information of stage 3(a). Note also that in the same bounds as those stated in the Theorem, we can find the biconnected components of the input digraph [20]. Finally, the sequential bound follows clearly by the parallel ones. ∎

4.2.2 The general case

The algorithm for detecting a negative cycle in a sparse digraph G is the following.

ALGORITHM Sparse_Neg_Cycle
BEGIN

1. Find a hammock-on-ears decomposition of G into $O(\tilde{\gamma})$ hammocks.

2. Detect if there is any negative cycle in some hammock. If a negative cycle is found in any hammock, then output one such cycle and stop.

3. Compress each hammock into an $O(1)$-sized graph such that the shortest paths between its attachment vertices are preserved and then compress G into a digraph of size $O(\tilde{\gamma})$. Examine if there is any negative cycle in this graph.

4. If a negative cycle is found, then output the cycle taking into account the subpaths contained in each hammock. Otherwise, output that there is no negative cycle in G.

END.

Theorem 4.3 *A negative cycle in an n-vertex sparse digraph G with real-valued edge costs, that can be decomposed into a minimum number of $O(\tilde{\gamma})$ hammocks, can be detected in $O(\log^2 n)$ time with $O(n + M_s(\tilde{\gamma}))$ CREW PRAM processors using $O(n + \tilde{\gamma}^2)$ space.*

Proof: The correctness of the algorithm as well as the space bound are clear. Note that step 1 needs $O(\log n \log\log n)$ time using $O(n)$ processors by Theorem 3.2. Step 2 needs $O(\log n \log^* n)$ time using $O(n/\log n \log^* n)$ CREW PRAM processors by Theorem 4.2. The implementation of step 3 is similar to the one described in step 4 of algorithm *Sparse_Apsp*. The compression of a hammock needs $O(\log^2 n)$ time with $O(n)$ processors [36]. The detection of a negative cycle in the compressed digraph is performed by runing the algorithm of [21], thus takes $O(\log^2 \tilde{\gamma})$ time and $O(M_s(\tilde{\gamma}))$ processors. Step 4 can be executed in $O(1)$ time using $O(n)$ processors. ■

Corollary 4.4 *A negative cycle in an n-vertex planar digraph G with real-valued edge costs, that can be decomposed into a minimum number of $O(\tilde{\gamma})$ hammocks, can be detected in $O(\log^2 n + \log^5 \tilde{\gamma})$ time with $O(n + \tilde{\gamma}^2 / \log^5 \tilde{\gamma})$ CREW PRAM processors using $O(n + \tilde{\gamma}^2)$ space. A sequential implementation of the above algorithm runs in $O(n + \tilde{\gamma}^{1.5} \log \tilde{\gamma})$ time.*

Proof: The parallel bounds are achieved, if in step 3 of algorithm *Sparse_Neg_Cycle* we run the algorithm of [4] (instead of the one in [21]). For the sequential implementation we have that: step 1 is performed in $O(n)$ time (by running a straightforward sequential implementation of our algorithm presented in Section 3). Step 2 needs $O(n)$ time by Theorem 4.2. The first part of step 3 (hammock compression) is described in [14] and needs $O(n)$ time. For the second part (detection of a negative cycle in the compressed version of G), we run the algorithm of [33] which needs $O(\tilde{\gamma}^{1.5} \log \tilde{\gamma})$ time. Finally, step 4 obviously runs in $O(n)$ time. ■

4.3 Computing all pairs reachability information

We treat the all pairs reachability problem as a degenerated version of the corresponding apsp problem. (Assign a cost of 1 for each arc $\langle v, w \rangle$ of G. If also there is no arc $\langle w, v \rangle$

in G, then add it with cost ∞. A vertex t is reachable by s iff their distance is not ∞.) Succinct encoding of reachability information can be stored into compact routing tables by defining for each arc $\langle v, w \rangle$ its compact label $R(v, w)$ as the set of vertices u such that there is a directed path from v to u with first edge $\langle v, w \rangle$. If we additionally want to output such paths, we must enforce a tie-breaking rule, to be applied when a vertex u belongs also to another set $R(v, z)$ and z is a neighbor of v. Therefore, we have the following.

Theorem 4.4 *Given an n-vertex sparse digraph G that can be decomposed into a minimum number of $O(\tilde{\gamma})$ hammocks, we can find on a CREW PRAM: (i) Apr information (encoded into compact routing tables) in $O(\log^2 n)$ time using $O(n\tilde{\gamma} + M_r(\tilde{\gamma}))$ processors and $O(n\tilde{\gamma})$ space; (ii) Apr information (using an alternative encoding) in $O(\log^2 n)$ time by employing $O(n + M_r(\tilde{\gamma}))$ processors and using $O(n + \tilde{\gamma}^2)$ space.*

Proof: Follows clearly by the above discussion, Theorem 4.1 and Corollary 4.2. ∎

Corollary 4.5 *Given an n-vertex planar digraph G that can be decomposed into a minimum number of $O(\tilde{\gamma})$ hammocks, we can find on a CREW PRAM: (i) Apr information (encoded into compact routing tables) in $O(\log^2 n + \log^5 \tilde{\gamma})$ time using $O(n\tilde{\gamma})$ processors and $O(n\tilde{\gamma})$ space; (ii) Apr information (using an alternative encoding) in $O(\log^2 n + \log^5 \tilde{\gamma})$ time by employing $O(n + \tilde{\gamma}^2 / \log^5 \tilde{\gamma})$ processors and using $O(n + \tilde{\gamma}^2)$ space.*

5 Conclusions and Further Results

We have presented here a technique, called hammock-on-ears decomposition, that decomposes any (di)graph into outerplanar subgraphs (called hammocks) satisfying certain separator properties. We have also shown that this technique is well-suited for parallel computation and can be advantageously used to improve the parallel bounds for all pairs shortest paths and related problems. Moreover, we managed to beat the transitive closure bottleneck (that appears to be inherent on these problems) for a significant class of graphs, namely that of sparse graphs. The interesting feature of the hammock-on-ears decomposition technique is that it can be used to design algorithms parameterized in terms of certain topological properties of the input (di)graph, i.e. in terms of $\tilde{\gamma}$. There are certain classes of graphs for which the value of $\tilde{\gamma}$ is small. Examples are: outerplanar graphs, graphs which satisfy the k-interval property [15] and graphs with genus $o(n)$. Also random $G_{n,p}$ graphs with edge probability $p = \Theta(1/n)$ seem to have a very small value for $\tilde{\gamma}$ [26].

We can achieve further improvements in the case where a digraph G is provided with an $O(n^{\mu})$-separator decomposition, $0 < \mu < 1$. Although it is known that a digraph with genus $0 < \gamma < n$ has a separator of size $O(\sqrt{n\gamma})$ [10, 17] and that such a separator can be computed in $O(n)$ time without an embedding of G to be provided [10], it is not known yet how to compute such a separator in NC. But there are some families of graphs for which an $O(n^{\mu})$-separator decomposition can be computed in NC. Consider for example the d-dimensional grid which has a trivial $O(n^{(d-1)/d})$-separator decomposition. Also in [35] it is shown that the class of k-overlap graphs embedded in d dimensions (which includes planar graphs) have a separator of size $O(k^{1/d}n^{(d-1)/d})$ that can be computed in NC. Hence, in the case where an $O(n^{\mu})$-separator decomposition is either provided with the input or it can be computed, we have the following results.

Corollary 5.1 *Let G be an n-vertex digraph with real-valued edge costs (but no negative cycles). Let also G be provided with an $O(n^{\mu})$-separator decomposition and can be decomposed into a minimum number of $O(\tilde{\gamma})$ hammocks. Then we can find the following on a CREW PRAM: (i) Apsp information (encoded into compact routing tables) in $O(\log^2 n + \log^3 \tilde{\gamma})$ time using $O(n\tilde{\gamma} + (\tilde{\gamma}^2 + \tilde{\gamma}^{2\mu+1})/\log^3 \tilde{\gamma})$ processors and $O(n\tilde{\gamma})$ space; (ii) Apsp information (using an alternative encoding) in $O(\log^2 n + \log^3 \tilde{\gamma})$ time by employing $O(n + (\tilde{\gamma}^2 + \tilde{\gamma}^{2\mu+1})/\log^3 \tilde{\gamma})$ processors and using $O(n + \tilde{\gamma}^2)$ space.*

Corollary 5.2 *Let G be an n-vertex digraph with real-valued edge costs. Assume that G is provided with an $O(n^{\mu})$-separator decomposition and that it can be decomposed into a minimum number of $O(\tilde{\gamma})$ hammocks. Then, there is an algorithm for detecting and outputting if it exists, a negative cycle in G in $O(\log^2 n + \log^3 \tilde{\gamma})$ time by employing $O(n + (\tilde{\gamma}^2 + \tilde{\gamma}^{2\mu+1})/\log^3 \tilde{\gamma})$ CREW PRAM processors. A sequential implementation of the above algorithm runs in $O(n + \min\{\tilde{\gamma}^{3\mu} + \tilde{\gamma}^{1+\mu}\log\tilde{\gamma}, \tilde{\gamma}^2\})$ time.*

Corollary 5.3 *Let G be an n-vertex digraph that can be decomposed into a minimum number of $O(\tilde{\gamma})$ hammocks. Let also G be provided with an $O(n^{\mu})$-separator decomposition. Then apr information can be computed in the same resource bounds with those given for apsp in Corollary 5.1.*

The hammock-on-ears decomposition technique can be also used to generalize the approach presented in [11] for planar digraphs and give efficient solution to the problem of making a time and space efficient preprocessing of a sparse digraph G such that a shortest path or distance query between any two vertices in G is answered very quickly. Moreover, the hammock-on-ears decomposition technique has recently used (along with other techniques) [12] to design efficient sequential and parallel algorithms for the dynamic version of the above problem, i.e. when the cost of an edge is modified or an edge is deleted.

We believe that the technique presented here has potential and will find applications to other problems, too.

Acknowledgements. We are grateful to Dimitris Sofotassios for pointing out the geometric approach to the intervals problem, and to Hristo Djidjev, Greg Frederickson, Christos Papadimitriou and Moti Yung for their insightful comments and their encouragement.

References

[1] K. Abrahamson, N. Dadoun, D. Kirkpatrick and T. Przytycka, "A Simple Parallel Tree Contraction Algorithm", *J. of Algorithms*, 10(1989), pp.287-302.

[2] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap, "Parallel Computational Geometry", *Algorithmica*, 3(3), 1988, 293-328.

[3] K.W. Chong and T.W. Lam, "Finding Connected Components in $O(\log n \log\log n)$ time on an EREW PRAM", *Proc. 4th ACM-SIAM Symp. on Discrete Algorithms*, 1993, pp.11-20.

[4] E. Cohen, "Efficient Parallel Shortest-paths in Digraphs with a Separator Decomposition", *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, 1993, pp.57-67.

[5] R. Cole, "Parallel Merge Sort", *Proc. 27th IEEE Symp. on FOCS*, 1986, pp.511-516.

[6] R. Cole and U. Vishkin, "Approximate Parallel Scheduling. Part I: The Basic Technique with Applications to Optimal Parallel List Ranking in Logarithmic Time", *SIAM J. Comp.*, Vol.17, No.1, February 1989, pp.128-142.

[7] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions", *J. of Symbolic Computations*, 9(3), 1990, pp.251-280.

[8] T. Cormen, C. Leiserson and R. Rivest, "Introduction to Algorithms", MIT Press & McGraw Hill, 1990.

[9] K. Diks, T. Hagerup and W. Rytter, "Optimal Parallel Algorithms for the Recognition and Colouring of Outerplanar Graphs", *Proc. MFCS 1988*, pp. 207-217, LNCS, Springer-Verlag.

[10] H. Djidjev, "A Linear Algorithm for Partitioning Graphs of Fixed Genus", SERDICA, Vol.11, 1895, pp.369-387.

[11] H. Djidjev, G. Pantziou and C. Zaroliagis, "Computing Shortest Paths and Distances in Planar Graphs", in *Proc. 18th ICALP*, 1991, LNCS, Vol. 510, pp. 327-339.

[12] H. Djidjev, G. Pantziou and C. Zaroliagis, "On-line and Dynamic Algorithms for Shortest Path Problems", MPI Technical Report, MPI-I-94-114, Saarbrücken, April 1994.

[13] G.N. Frederickson, "Using Cellular Graph Embeddings in Solving All Pairs Shortest Path Problems", *Proc. 30th Annual IEEE Symp. on FOCS*, 1989, pp.448-453; also CSD–TR-897, Purdue University, August 1989.

[14] G.N. Frederickson, "Planar Graph Decomposition and All Pairs Shortest Paths", *J. ACM*, Vol.38, No.1, January 1991, pp.162-204.

[15] G.N. Frederickson and R. Janardan, "Designing Networks with Compact Routing Tables", *Algorithmica*, 3(1988), pp.171-190.

[16] H. Gazit and G. Miller, "A deterministic parallel algorithm for finding a separator in planar graphs", Technical Report CMU-CS-91-103, Carnegie-Mellon University, 1991.

[17] J. Gilbert, J. Hutchinson and R. Tarjan, "A Separator Theorem for Graphs of Bounded Genus", *Journal of Algorithms*, 5 (1984), pp.391-407.

[18] M. Goodrich, "Intersecting Line Segments in Parallel with an Output-Sensitive Number of Processors", *Proc. 1st Symposium on Parallel Algorithms and Architectures*, June 1989, pp.127-136.

[19] J.L. Gross and T.W. Tucker, "Topological Graph Theory", John Wiley, New York, 1987.

[20] T. Hagerup, "Optimal Parallel Algorithms for Planar Graphs", *Information and Computation*, 84 (1990), pp.71-96.

[21] Y. Han, V. Pan and J. Reif, "Efficient Parallel Algorithms for Computing All Pair Shortest Paths in Directed Graphs", *Proc. 4th ACM Symp. on Parallel Algorithms and Architectures*, 1992, pp.353-362.

[22] R. Hassin, "Maximum Flow in (s,t) planar networks", *Inform. Proc. Letters*, 13 (1981), pp.107.

[23] J. JáJá, "An Introduction to Parallel Algorithms", Addison-Wesley, 1992.

[24] M.Y. Kao and P. Klein, "Towards Overcoming the Transitive-Closure bottleneck: Efficient Parallel Algorithms for Planar Digraphs", *Proc. of 22nd ACM STOC*, 1990, pp.181-192.

[25] R. Karp and V. Ramachandran, "Parallel Algorithms for Shared-Memory Machines", *Handbook of Theoretical Computer Science*, Ed. J. van Leeuwen, pp.869-941, 1990, Elsevier Science Publishers.

[26] D. Kavvadias, G. Pantziou, P. Spirakis and C. Zaroliagis, "Efficient Sequential and Parallel Algorithms for the Negative Cycle Problem", to appear in *Proc. ISAAC'94*, LNCS, Springer-Verlag, 1994.

[27] E.L. Lawler, "Combinatorial Optimization: Networks and Matroids", Holt, Rinehart and Winston, 1976.

[28] T. Lengauer, "Efficient Algorithms for the Constraint Generation and Intergrated Circuit Layout Compaction", *Proc. of 9th Workshop on Graph-Theoretic Concepts in Computer Science (WG83)*, pp.219-230, 1983.

[29] D. Maier, "An Efficient Method for Storing Ancestor Information in Trees", *SIAM J. on Comp.*, Vol.8, N.4, pp.599-618, 1979.

[30] Y. Maon, B. Schieber and U. Vishkin, "Parallel ear decomposition search (EDS) and st-numbering in graphs", *Theoretical Computer Science*, 47 (1986), pp.277-298.

[31] E.M. McCreight, "Priority Search Trees", *SIAM Journal on Computing*, No.14, 1985, 257-276.

[32] K. Mehlhorn, "Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry", Springer-Verlag, 1984.

[33] K. Mehlhorn and B. Schmidt, "A Single Source Shortest Path Algorithm for Graphs with Separators", in *Proc. FCT83*, LNCS 158, pp.302-309, 1983, Springer-Verlag.

[34] G. Miller and J. Naor, "Flows in Planar Graphs with Multiple Sources and Sinks", *Proc. 31st IEEE Symp. on FOCS*, 1989, pp.112-117.

[35] G. Miller, S-H Teng and S. Vavasis, "A unified geometric approach to graph separators", *Proc. 32nd IEEE Symp. on FOCS*, 1991, pp.538-547.

[36] G. Pantziou, P. Spirakis and C. Zaroliagis, "Efficient Parallel Algorithms for Shortest Paths in Planar Digraphs", *BIT* 32 (1992), pp.215-236.

[37] C.H. Papadimitriou and K. Steiglitz, "Combinatorial Optimization: Algorithms and Complexity", Prentice-Hall, 1982.

[38] G. Ramalingan and T. Reps, "On the Computational Complexity of Incremental Algorithms", Technical Report, University of Wisconsin-Madison, 1991.

[39] T. Reps, M. Sagiv and S. Horwitz, "Interprocedural Dataflow Analysis via Graph Reachability", Technical Report TR 94-14, Datalogisk Institut, University of Copenhagen, 1994.

[40] Y. Shiloach and U. Vishkin, "An $O(\log n)$ parallel connectivity algorithm", *J. of Algorithms*, Vol.3, 1982, pp.57-67.

[41] P. Spirakis and A. Tsakalidis, "A Very Fast, Practical Algorithm for Finding a Negative Cycle in a Digraph", in *Proc. of 13th ICALP*, pp. 397-406, 1986.

[42] R.E. Tarjan, "Data Structures and Network Algorithms", SIAM, 1983.

[43] C. Thomassen, "The graph genus problem is NP-complete", *J. of Algorithms*, 10 m(1989), pp.568-576.

[44] D. Yellin and R. Strom, "INC: A language for incremental computations", *ACM Trans. Prog. Lang. Systems*, 13 (2), pp.211-236, April 1991.

[45] J. van Leeuwen and R. Tan, "Computer Networks with compact routing tables", in *The Book of L*, G. Rozenberg and A. Salomaa (eds.), Springer-Verlag, NY (1986), pp.259-273.

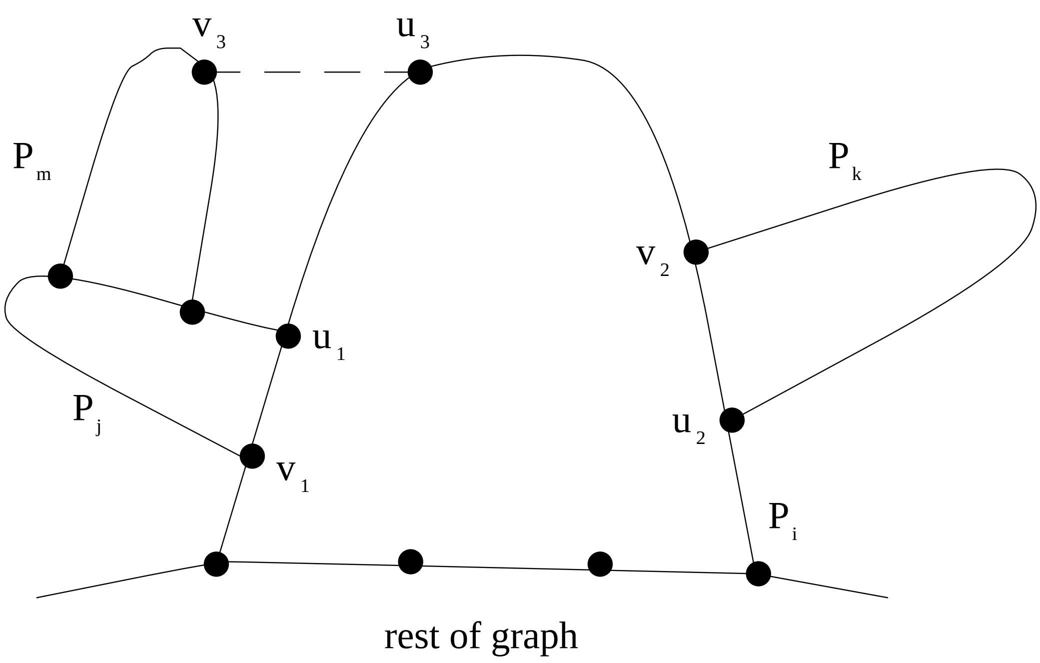

Fig 3.1: m>j>i, k>i

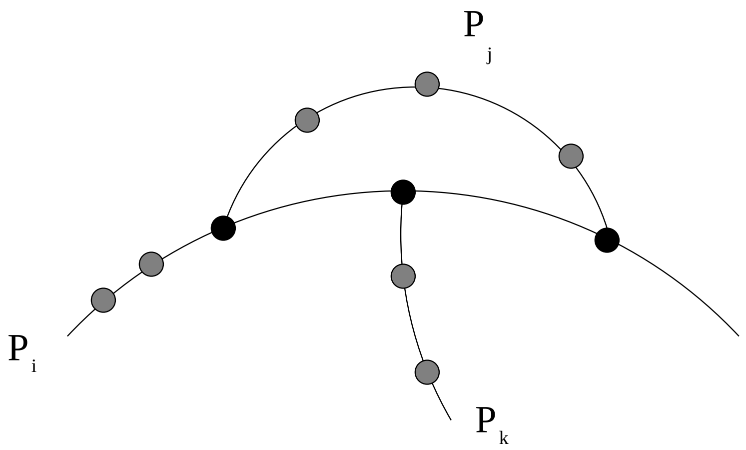

Fig 3.2: ● = dividing vertex of stage 1

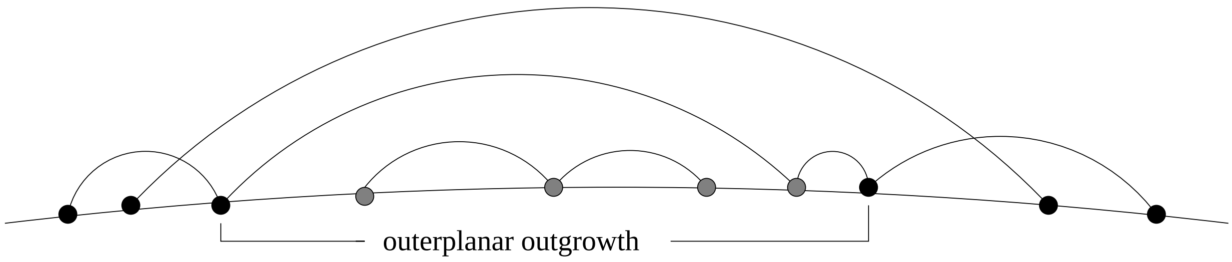

Fig 3.3: ● = dividing vertex of stage 2

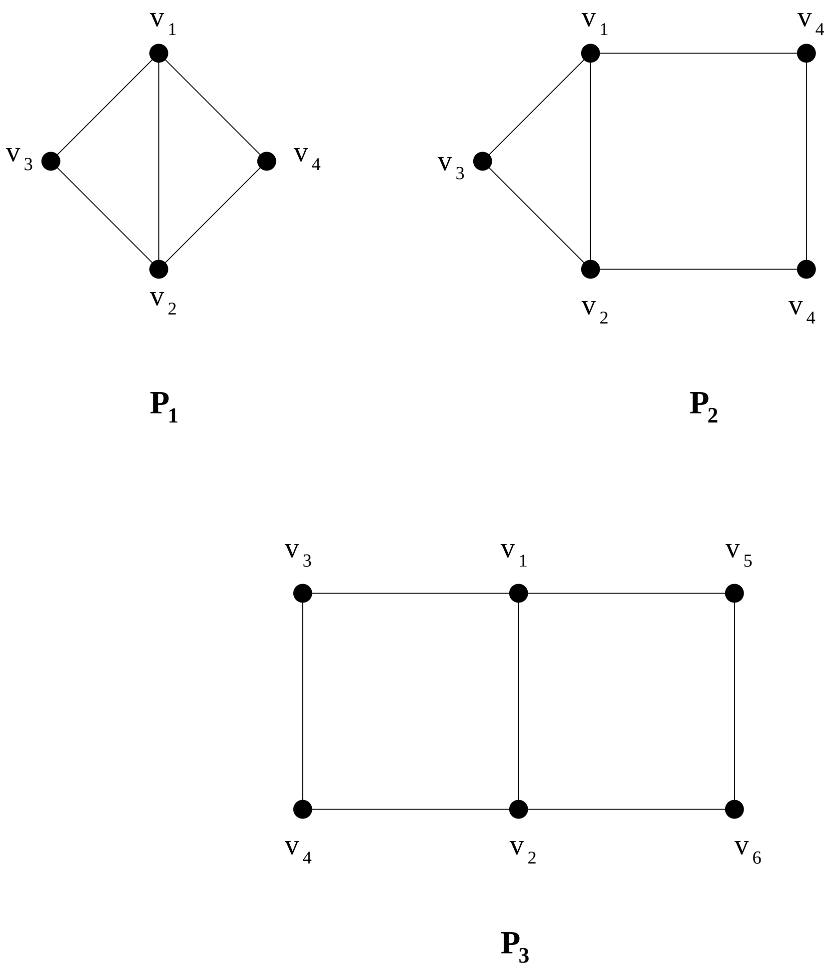

Fig 3.4.

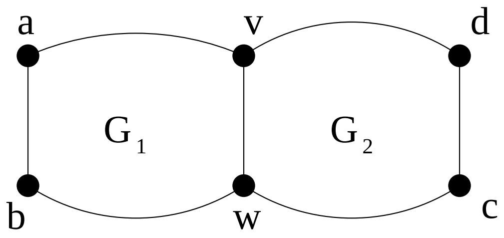

Fig 4.1: Lemma 4.1.

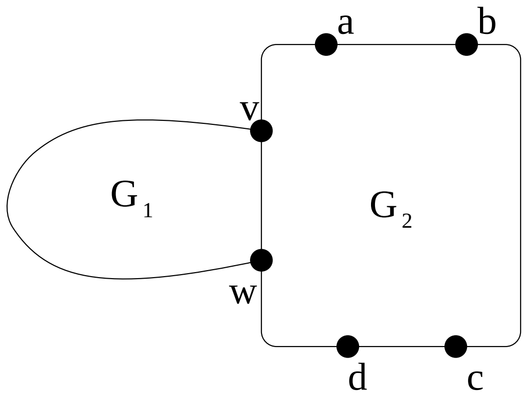

Fig 4.2: Lemma 4.2.

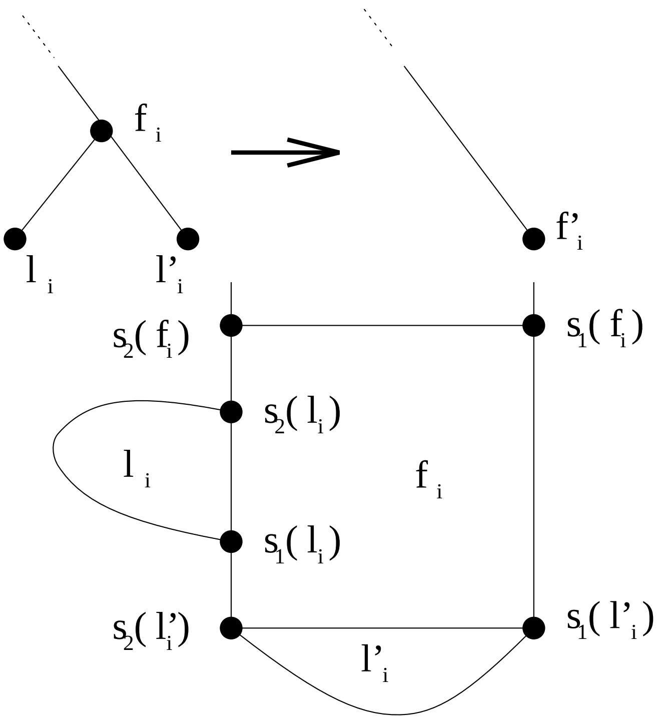

Fig 4.3: SHUNT operation, TYPE-1.

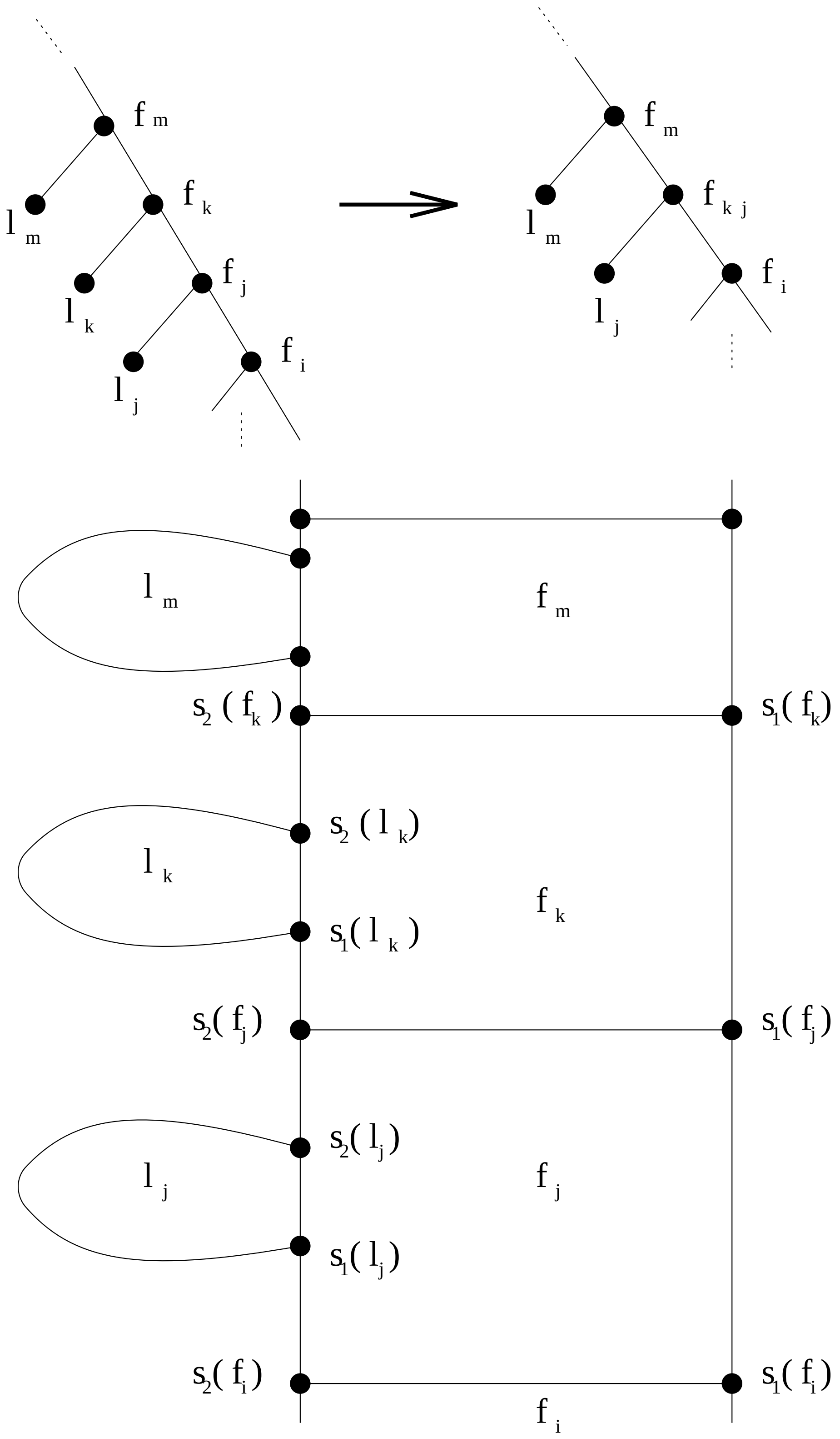

Fig 4.4: SHUNT operation, TYPE-2.

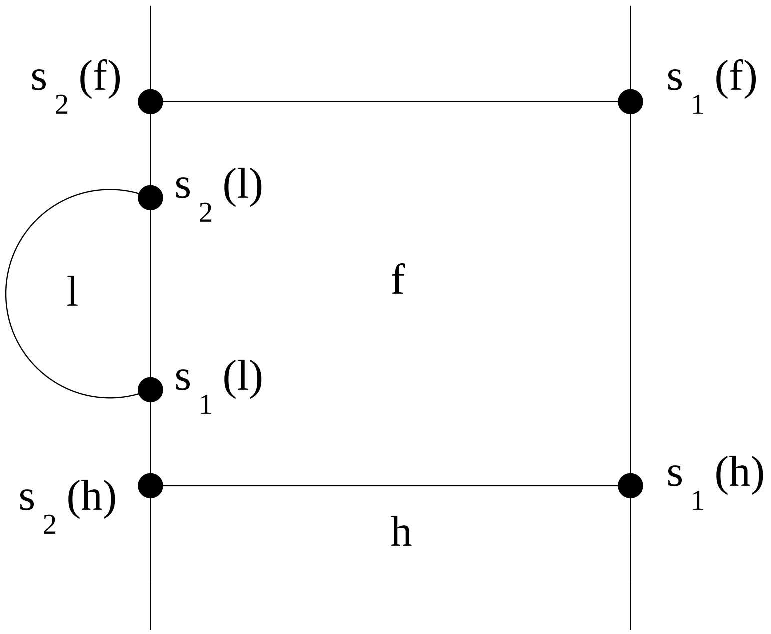

Fig 4.5.