Hammock-on-Ears Decomposition: A Technique for the Efficient Parallel Solution of Shortest Paths and Other Problems* (to appear in MFCS'94)

Dimitris Kavvadias[1,2], Grammati E. Pantziou[1,4],
Paul G. Spirakis[1,3] and Christos D. Zaroliagis[1,5]

[1] Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
[2] Dept of Mathematics, Patras University, 26500 Patras, Greece
[3] Dept of Computer Sc. & Eng., Patras University, 26500 Patras, Greece
[4] Dept of Math. & Computer Sc., Dartmouth College, Hanover NH 03755, USA
[5] Max-Planck Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany

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. 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. This decomposition provides a general framework for solving graph problems efficiently in parallel. Its value is demonstrated by using it to improve previous bounds for shortest paths and related problems in the class of sparse graphs (which includes planar and bounded genus graphs).

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, sorting, depth-first search, etc [15]. 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 [4]). 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*) that satisfy certain separator conditions. 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 [7, 8] and on the well-known ear decomposition technique [15, 19], and non-trivially extends previous work for planar digraphs [21] to any digraph. We demonstrate its applicability by using it to improve the parallel bounds for certain path problems in a

* Supported by the EEC ESPRIT Basic Research Action No. 7141 (ALCOM II), by the Ministry of Education of Greece and by the NSF grant No. CDA-9211155. Email: kavadias/spirakis@cti.gr, pantziou@cs.dartmouth.edu, zaro@mpi-sb.mpg.de.

significant and quite large 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 with genus bounded by any function $\gamma(n) = o(n)$.

The hammock-on-ears 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.

The hammock-on-ears decomposition technique provides a *general scheme* for solving problems, which 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.

We apply this scheme to the all pairs shortest paths (apsp) and all pairs reachability (apr) problems (for definitions and applications see [3, 6–8, 15, 21, 23]), and achieve considerable improvements in the case of sparse digraphs. For other related applications using the above scheme, see [18].

Overview of previous work and motivation. The sequential hammock decomposition technique was developed by Frederickson in [7, 8]. 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 [8] to be $\tilde{\gamma} = \tilde{\gamma}(G) = \Theta(\gamma(G')) = \Theta(\bar{\gamma}(G'))$ where $\gamma(G')$ and $\bar{\gamma}(G')$ are the genus and crosscap number [13] 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[6]. Therefore, $\tilde{\gamma}(G)$ can range from 1 up to $\Theta(m)$ depending on the topology of the graph. The decomposition is achieved in $O(n + m)$ time.

[6] 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 minumum number of faces that together cover all vertices of G. Note that q ranges from 1 up to $\Theta(n)$.

Hammock decomposition seems to be an important topological decomposition of a graph which proved useful in solving efficiently apsp problems [7, 8, 21], along with the idea of storing shortest path information into compact routing tables[7] [9, 25]. Compact routing tables are very useful in space-efficient methods for message routing in distributed networks [9, 25]. 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 [7, 8]. 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 parallel time $O(\log^2 n)$ on a CREW PRAM), and hence beat the so-called *transitive closure bottleneck* [16]. Such a "beating", for planar digraphs, was first achieved in [21]. (The best value, up to now, for $M_s(n)$ is $O(n^3(\log\log n)^{1/3}/(\log n)^{7/6})$ [14].) If the digraph is provided with an $O(n^\mu)$-separator decomposition, $0 < \mu < 1$, it has been recently shown by Cohen [3] that apsp can be computed in $O(\log^3 n)$ time and $O(n^{3\mu}+n^2+n^{2\mu+1})$ work on an EREW PRAM. For the case of planar digraphs, where an $O(n^{1/2})$-separator decomposition can be computed using the algorithm of [11], the results in [3] imply an $O(\log^5 n)$-time, $O(n^2)$-work CREW PRAM algorithm for the apsp problem. For previous work and applications of the apr problem see [3, 15].

Our results. Given any (di)graph 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 by the input. (An implementation 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 [8].) Using the above result, as well as the general scheme discussed before, we give an algorithm for computing apsp information (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 or digraphs with a small separator (e.g. graphs with genus $o(n)$), 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[8] 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 [6–8], 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 or has a small separator, the processor bound can be further reduced to $O(n+\tilde{\gamma}^2/\log^5 \tilde{\gamma})$. Similar results are obtained for the apr problem (Section 4).

We note that: (i) the hammock-on-ears decomposition fully extends the topological characteristics of the input digraph. The better these characteristics are

[7] 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.

[8] The best sequential result for planar digraphs [7] achieves $O(n\tilde{\gamma})$ time and space.

(i.e. the smaller the $\tilde{\gamma}$), the more efficient the algorithms for the above applications become; (ii) our algorithms for all the above applications explicitly construct the graph decomposition.

2 Preliminaries

A graph is called *outerplanar* if it can be embedded in the plane so that all the vertices are on one face. Note that if G is outerplanar then $\tilde{\gamma} = 1$, since $q = 1$ (all vertices are covered by one face) and $\gamma(G) = 0$. 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 [8]. 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. G' is $G + v$, together with an arc from v to each vertex of 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 at most four attachment vertices per hammock. 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. Let $\tilde{\gamma}$ 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 [24]), we follow here the approach used in [8], 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 $O(\tilde{\gamma})$ *partial* hammocks using two operations: pseudo-absorption and pseudo-sequencing. As their names indicate these operations are analogous to absorption and sequencing 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 Hammock-on-Ears Decomposition Technique

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 [8].

Lemma 1 ([8]). *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*. 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*, 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.

The first step in the *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 vertices 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:

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 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.

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

Lemma 2. *Each non-trivial ear produced from stage 1 of 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.*

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 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 vertices 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:

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.

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

Lemma 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.*

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 $O(\log m)$ by employing $O(m)$ CREW PRAM processors [17] where m is the number of intervals involved, using a data structure called "priority search tree" [20]. We note here that the literature of parallel geometry is very rich in related problems that study intersections of line segments in the plane (see e.g. [1, 12]). 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.

Theorem 1. *Algorithm Find_Outgrowths correctly identifies all outerplanar outgrowths of a graph $G = (V, E)$ ($|V| = n$, $|E| = m$) 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: Correctness of the algorithm comes from lemmata 2 and 3 (whose proofs are given in the full paper [17]). The resource bounds come from the above discussion and the bounds of the parallel connected components algorithm [2, 19, 22]. ∎

The parallelization of the *pseudo sequencing* procedure is based on the following idea. 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 discussed in [17].) Now observe that since every vertex 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. Using a similar idea with [8], we can prove that any sequence of 7 faces in a hammock has a constant number of different forms which can be identified by a single processor in $O(1)$ time after performing some local degree tests. Therefore, 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. Since each contraction joins two neighboring faces in one, $\frac{1}{7}$th of a hammock is contracted in every step. Therefore, in $O(\log n)$ time using $O(n+m)$ CREW PRAM processors we can generate a labeling of the edges of the graph that gives a partial hammock decomposition. Combining all the results in this section we get:

Theorem 2. *Given a digraph $G = (V, E)$ ($|V| = n$, $|E| = m$), the decomposition of G into $O(\tilde{\gamma})$ 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.*

4 Applications

Recall the general scheme discussed in the introduction. The algorithm for computing apsp information in a sparse digraph G proceeds as follows: (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 partial hammock. (4) Find apsp in G between each pair of attachment vertices of partial hammocks. This is done by first replacing each hammock H by a digraph $C(H)$ of $O(1)$ size. $C(H)$ is just a complete digraph on the attachment vertices in H, with the cost of each edge being the shortest distance in H between its endpoints. In this way, a new digraph $C(G)$ is created which is of size $O(\tilde{\gamma})$. Find apsp in $C(G)$ by running the algorithm of [14]. (5) For each pair of partial hammocks determine succinct shortest paths information for each vertex in one partial hammock to all vertices in the other partial hammock. (6) For each partial 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.)

Theorem 3. *Given a sparse digraph G (with real-valued edge costs but no negative cycles), we can find: (i) Apsp information (encoded in compact routing tables) in $O(\log^2 n)$ time using $O(n\tilde{\gamma} + M_s(\tilde{\gamma}))$ CREW PRAM processors and $O(n\tilde{\gamma})$ space. (ii) Apsp information (using an alternative encoding) in $O(\log^2 n)$ time using $O(n + M_s(\tilde{\gamma}))$ CREW PRAM processors and $O(n + \tilde{\gamma}^2)$ space.*

Proof: The bounds of step 1 come from theorem 5. 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 [21]. Step 4 needs $O(\log n)$ time with $O(n/\log n)$ processors for computing the edge costs of $C(H)$ [21], and $O(\log^2 \tilde{\gamma})$ time using $M_s(\tilde{\gamma})$ processors for apsp in $C(G)$. Step 5 needs for all hammocks $O(\log n)$ time using $O(n\tilde{\gamma})$ processors by [21]. Finally, step 6 (again by [21]) needs $O(\log^2 n)$ time using $O(n)$ processors. If we use the

alternative encoding for apsp information we actually do not need to run steps 5 and 6. The space bounds follow from the discussion in the introduction. ∎

In the special case of planar digraphs (or digraphs with a small separator), we use the algorithm of [3] (instead of the one in [14]) in step 4 of our algorithm. The apr problem is handled as a degenerated version of the corresponding apsp problem. In the case of planar digraphs (or digraphs with small separators) the processor bound can be reduced to $O(n + \tilde{\gamma}^{1.19})$, using the result of [5] and the alternating encoding for storing reachability information. Note that a negative cycle in the input digraph G can be detected within the same bounds as those stated in part (ii) of theorem 6 (see [18]). The processor bound can be also reduced to $O(n + \tilde{\gamma}^2/\log^5 \tilde{\gamma})$ in the special case of planar digraphs or digraphs with a small separator [18].

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. A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap, "Parallel Computational Geometry", *Algorithmica*, 3(3), 1988, 293-328.
2. 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 SODA* , 1993, pp.11-20.
3. E. Cohen, "Efficient Parallel Shortest-paths in Digraphs with a Separator Decomposition", *Proc. 5th ACM SPAA*, 1993, pp.57-67.
4. R. Cole, "Parallel Merge Sort", *Proc. 27th IEEE FOCS*, 1986, pp.511-516.
5. D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions", *J. of Symbolic Computations*, 9(3), 1990, pp.251-280.
6. 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.
7. G.N. Frederickson, "Planar Graph Decomposition and All Pairs Shortest Paths", *J. ACM*, Vol.38, No.1, January 1991, pp.162-204.
8. G.N. Frederickson, "Using Cellular Graph Embeddings in Solving All Pairs Shortest Path Problems", *Proc. 30th Annual IEEE FOCS*, 1989, pp.448-453.
9. G.N. Frederickson and R. Janardan, "Designing Networks with Compact Routing Tables", *Algorithmica*, 3(1988), pp.171-190.
10. M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms", *JACM*, 34(1987), pp. 596-615.
11. H. Gazit and G. Miller, "A deterministic parallel algorithm for finding a separator in planar graphs", Report CMU-CS-91-103, Carnegie-Mellon University, 1991.
12. M. Goodrich, "Intersecting Line Segments in Parallel with an Output-Sensitive Number of Processors", *Proc. 1st ACM SPAA*, June 1989, pp.127-136.
13. J.L. Gross and T.W. Tucker, "Topological Graph Theory", John Wiley, 1987.
14. Y. Han, V. Pan and J. Reif, "Efficient Parallel Algorithms for Computing All Pair Shortest Paths in Directed Graphs", *Proc. 4th ACM SPAA*, 1992, pp.353-362.
15. J. JáJá, "An Introduction to Parallel Algorithms", Addison-Wesley, 1992.
16. 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.

17. D. Kavvadias, G. Pantziou, P. Spirakis and C. Zaroliagis, "Hammock-on-Ears Decomposition: A Technique for Parallel and On-line Path Problems", CTI Technical Report, TR-93.05.22, Computer Technology Institute, Patras, 1993.
18. D. Kavvadias, G. Pantziou, P. Spirakis and C. Zaroliagis, "Efficient Sequential and Parallel Algorithms for the Negative Cycle Problem", *Proc. 5th ISAAC'94*, LNCS, to appear.
19. 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.
20. E.M. McCreight, "Priority Search Trees", *SIAM Journal on Computing*, No.14, 1985, 257-276.
21. G. Pantziou, P. Spirakis and C. Zaroliagis, "Efficient Parallel Algorithms for Shortest Paths in Planar Digraphs", *BIT* 32 (1992), pp.215-236.
22. Y. Shiloach and U. Vishkin, "An $O(\log n)$ parallel connectivity algorithm", *J. of Algorithms*, Vol.3, 1982, pp.57-67.
23. R.E. Tarjan, "Data Structures and Network Algorithms", SIAM, 1983.
24. C. Thomassen, "The graph genus problem is NP-complete", *J. of Algorithms*, 10(1989), pp.568-576.
25. J. van Leeuwen and R. Tan, "Computer Networks with compact routing tables", *The Book of L*, G. Rozenberg and A. Salomaa (eds.), Springer, 1986, pp.259-273.