TECHNICAL REPORT NO. 335

New Divide-and-Conquer Techniques For Finding Disjoint Paths

by

Fang Wan

September 1991

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY
Bloomington, Indiana 47405-4101

New Divide-and-Conquer Techniques for Finding Disjoint Paths

Fang Wan*
Department of Computer Science
Indiana University
Bloomington, Indiana 47405

September 9, 1991

Abstract

This paper presents the first parallel algorithm that finds maximal vertex-disjoint paths (MVDP) between sources and sinks in an undirected planar graph using poly-logarithm time and a linear number of processors on the parallel random access machine (PRAM). We also reduce the MVDP problem to the depth-first search (DFS) problem for the first time. The reduction treats a graph as a *tripartite* graph formed by sources, sinks, and maximal connected components of intermediate vertices, and successively uses a novel Divide-and-Conquer technique based on cycle separators to decompose those maximal connected components into smaller components. Consequently we can construct a DNC MVDP algorithm for graphs whose maximal connected components of intermediate vertices are planar, although no DNC DFS algorithm for this class of non-planar graphs is known.

1 Introduction

Consider the following graph problem in a graph of n vertices: given k *sources* and k *sinks* that are one-to-one matched by labels, find a maximal set of vertex-disjoint simple paths that are from sources to matching sinks via *intermediate vertices* which are neither sources nor sinks, where k can be any function of n. We call this problem as *maximal s-t paths* problem or *S-T* problem for short. Another important problem, called *maximal vertex-disjoint paths* problem or *MVDP* problem, is finding a maximal set of vertex-disjoint simple paths that are from sources to sinks, not necessarily matched, via intermediate vertices. In this paper the two problems are sometimes all referred as the MVDP problem when their difference is not an issue. The MVDP problem is interesting in its own right for its role in problems such as graph connectivity, message routine on networks, and VLSI layout. MVDP is also an approximate solution of the maximum vertex-disjoint paths problem which is NP-complete.

*This research has been supported in part by NSF Grant CCR-89-09535

A *DNC* algorithm is a *deterministic* parallel algorithm using *poly-logarithm* time and a *polynomial* number of processors on the parallel random access machine (PRAM) [14], where complexities are functions of the size of the algorithm's input. All computation complexities in this paper are measured by the concurrent-read concurrent-write (CRCW) PRAM.

Although we can easily construct a linear-time sequential MVDP algorithm using standard sequential techniques [3], previously there was no DNC MVDP algorithm for any non-trivial class of graphs. Phenomena of this kind reveal difficulties involved in finding fast parallel algorithms for some fundamental linear-time sequential algorithms. Evidently we need new parallel techniques and a better understanding of the problem.

For the important case of undirected planar graphs, we present an algorithm which uses $O(\log^3 n)$ time and n processors for the S-T problem and use $O(\log^4 n)$ time and n processors for the MVDP problem. Its speedup and its processor-time product are all optimal within a poly-logarithm factor. For undirected graphs in general, we use a new Divide-and-Conquer technique to reduce the MVDP problem to the depth-first search (DFS) problem for the first time. However we can use the same reduction to construct a DNC MVDP algorithm for a class of non-planar graphs while no DNC DFS algorithm for the same class of graphs is known. This potential gap between the MVDP problem and the DFS problem is not known before.

One major obstacle to a DNC MVDP algorithm is that paths found independently may joint each other so that few paths can survive in the worst case. The MVDP problem differs from other 'maximal set' problems such as maximal independent set and maximal matching in which independent local solutions can produce a constant portion of a final maximal set. Thus we develop a new paradigm which employs a novel Divide-and-Conquer technique making use of *cycle separators*. Instead of trying to find a substantial amount of *eligible* paths and accumulate them into a maximal set, we delete some eligible paths to decompose a given graph and iterate the process until we get a simplified graph in which we know how to find MVDP efficiently.

More specifically, first we develop a DNC MVDP algorithm for *tripartite graphs* whose intermediate vertices are separated from each other by sources and sinks and which is the target of our simplification process. Then we treat maximal connected components of intermediate vertices as pseudo vertices so a graph looks like a tripartite graph. Given cycle separators in these maximal connected components, we find eligible paths going through the cycle separators and delete them so that the maximum size of remain maximal connected components of intermediate vertices is a constant factor smaller. Within logarithmic iterations of simplification, a graph is cut into a tripartite graph by a set of eligible paths. The union of MVDP in the remain tripartite graph and paths deleted in the simplification process is MVDP of the original graph. Since the DFS problem and the cycle separator problem are DNC equivalent [24, 12], the MVDP problem is reduced to the DFS problem. Based on DNC DFS algorithms for undirected planar graphs [22], we claim a DNC MVDP algorithm for undirected graphs in which maximal connected components of intermediate vertices are planar. No DNC DFS algorithm for the same class of graphs is known.

The above paradigm has no straight forward efficient implementation for undirected planar graphs. Thus we develop additional algorithms which find MVDP on a tripartite planar graph in

$O(\log^2 n)$ time and find *priority maximal matching* on a *priority bipartite planar graph* in $O(\log^2 n)$ time using n processors. They improve the general paradigm so that it finds S-T and MVDP in an undirected planar graph in $O(\log^3 n)$ time and $O(\log^4 n)$ time respectively using n processors.

Previous results [1, 8, 2] imply a randomized NC (RNC) algorithm for the MVDP problem, but there was no DNC MVDP algorithm for planar graphs before. Goldberg, Plotkin, and Vaidya developed a parallel algorithm for a generalized MVDP problem [8]. But their algorithm is not a DNC algorithm. The MVDP problem is a special case of their generalized MVDP problem. Given k larger than 2, finding maximum vertex-disjoint paths is an NP-complete problem except when it uses $n^{k^{k^{\cdots^k}}\}k}$ time in a planar graph of fixed k [15, 18, 6, 21, 17, 16]. MVDP is a useful approximation of maximum vertex-disjoint paths when no better approximation is known.

DFS is of fundamental importance in computational graph theory and has many applications. Finding a DNC DFS algorithm for general graphs has been origins of many research works. So far, all efforts have failed because DFS is a very sequential precess [20, 4, 1, 8]. In fact, the problem of finding the very DFS tree found by a sequential greedy algorithm is P-complete [20]. However, an arbitrary DFS tree may be found in general graphs by RNC algorithms [1, 2] and in planar graphs by DNC algorithms [24, 22, 12, 13]. A very important technique developed by the previous efforts is reducing the DFS problem to a generalized MVDP problem [1, 8, 2]. But it remains unknown whether the two problems are DNC equivalent. As a partial answer, our result implies that the MVDP problem (a special case of the generalized MVDP problem) can be reduced to the DFS problem. Whether DFS can be reduced to MVDP is open.

In Section 2 we introduce a paradigm for solving the maximal vertex-disjoint paths problems and implement the paradigm using DNC subroutines except one used to find cycle separators. In Section 3 we present linear-processor DNC maximal vertex-disjoint paths algorithms for undirected planar graphs.

2 Find vertex-disjoint paths in parallel

We assume, without lose of generality, that every undirected graph $G = (V, E)$ has k sources $\{s_1, \cdots, s_k\} = S$ and k sinks $\{t_1, \cdots, t_k\} = T$ so that $S \subseteq V$, $T \subseteq V$, $S \cap T = \emptyset$ and $S \times T \cap E = \emptyset$. Otherwise, dummy sources and sinks can be added. A source and a sink that have the same subscript are said to be matched. Intermediate vertices, denoted as $R = V - S - T$, are called R-vertices. A planar graph is a graph can be drawn on a plane without edges crossing each other. $G = (S \cup R \cup T, E)$ is bipartite if $R = \emptyset$ and is tripartite if $E \subseteq (S \times R \cup R \times T)$. We also use notations $|V| = n$, $|E| = m$, $|S| = n_s$ and $|T| = n_t$. $\Delta(v)$ is the degree of vertex v. An edge is called a *dangling* edge if it only incidents to one vertex whose degree is two or larger.

For the purpose of this paper, an eligible path is referred to a simple path which connects a source and a sink via R-vertices. The source and the sink are matched in the S-T problem but not necessarily in the MVDP problem. A *Pseudo-R-vertex* of G is a maximal connected component of R-vertices and is called a R^*-vertex. $G^* = (S \cup R^* \cup T, E)$ is a tripartite graph constructed from $G = (S \cup R \cup T, E)$ by contracting each R^*-vertex into one vertex.

2.1 Find vertex-disjoint paths in tripartite graphs

First we solve the MVDP problem in tripartite graphs. The underline principle obtained is also important for general graphs if we recall that a general graph G looks like a tripartite graph when it is viewed as G^*. Therefore, what we do for tripartite graphs in this section will later be applied to general graphs after proper modifications.

Lemma 1 *Maximal s-t paths in a tripartite graph can be found in $O(\log^3 n)$ time with $n+m$ processors.*

Proof: Given any tripartite graph G, an edge (s_i, r) is redundant if (r, t_i) is not in G and verse versa, where $1 \leq i \leq k$. In parallel on each R-vertex r of G, we label and sort edges linking r to sources by source subscripts. A processor is then assigned to each edge linking r to a sink t_i to look up in the sorted edges linking r to sources an edge linking r to s_i, and mark the two edges if it succeeds. All unmarked edges are redundant and can be deleted from G because they are on any eligible path. The only non-trivial computation involved in removing redundant edges is sorting n elements in $O(\log n)$ time with linear-processor [5]. Therefore we can use $O(\log n)$ time and $n+m$ processors to remove all redundant edges from G and obtain G'. Finding maximal s-t paths in G' is as easy as finding a maximal matching on a bipartite graph. We first find a maximal matching in a bipartite graph formed by sources of G' and R-vertices of G' and then extend an edge (s_i, r) in the maximal matching with an edge (r, t_i) which is guaranteed to exist after our simplification process. So a maximal matching between S and R of G' decides a set of vertex-disjoint eligible paths and the set is maximal. Otherwise, it contradicts either the assumption of G' or the definition of maximal matching. The best maximal matching algorithm for general graphs uses $O(\log^3 n)$ time and $n+m$ processors [11]. ∎

Lemma 2 *Maximal vertex-disjoint paths in a tripartite graph can be found in $O(\log^4 n)$ time with n^5 processors.*

Proof: Given a tripartite graph G, we construct a graph G' in which vertices represent distinct eligible paths in G and two vertices share an edge if and only if the two corresponding eligible paths in G share at least a vertex. G' may have up to n^3 vertices and only up to n^5 edges because a vertex of G is on at most n^2 distinct eligible paths in G. Obviously there is an one-to- one mapping between distinct maximal independent sets of G' and distinct sets of maximal vertex-disjoint paths of G. Finding a maximal independent set in G' uses $O(\log^4 n)$ time and n^5 processors on an EREW PRAM when a linear-processor MIS algorithm for general graphs is used [9]. ∎

Proofs of the above lemmas are based on finding a maximal independent set among all possible eligible paths. A much more efficient linear-processor algorithm will be developed for planar graphs in a later section. How to improve the MVDP algorithm for general tripartite problem is open problem.

2.2 A paradigm for finding vertex-disjoint paths

Our strategy is trying to simplify a graph into a tripartite graph first because we have a DNC MVDP algorithm for tripartite graphs now. The key simplification technique is finding and removing vertex-disjoint eligible paths which break R^*-vertices into components that are a constant factor smaller. Using such a technique $O(\log n)$ times, we can cut a graph into a tripartite graph. This idea leads to the following paradigm.

Algorithm 1 *Find-Vertex-Disjoint-Paths*

Input: An undirected graph, G
Output: PATHS that contains S-T (MVDP) of G

1. **While** the current graph is not tripartite **do**
   Find a cycle separator in each non-trivial maximal connected component of R-vertices and a set, P, of disjoint eligible paths so that vertices on the cycle separators either are on these paths or are not on any additional eligible paths. Add P to PATHS and delete P and all cycle separators from the current G. { a comment: the maximal size of R^*-vertices decreases a constant factor in each iteration. }

2. Find S-T (MVDP) on the current tripartite graph and return them with the vertex-disjoint paths found in the first step.

A cycle (path) separator in a graph is a simple cycle (path) whose removal divides the graph into connected components smaller than two third of the original graph. Obviously Algorithm 1 runs at most $O(\log n)$ iterations in the first step and has a DNC subroutine for the second step by Lemma 1 and Lemma 2. We will prove that vertex-disjoint eligible paths required in the first step of Algorithm 1 exist and can be found by a DNC algorithm when cycle separators are given. Thus we show that the MVDP problem can be DNC reduced to the cycle separator problem. We will assume that a cycle separator algorithm uses $T_{sep}(n)$ time and $P_{sep}(n)$ processors.

2.3 How to break a R^*-vertex with eligible paths

In this section we show how to break a R^*-vertex r^* by removing some disjoint eligible paths passing through a cycle separator of r^*. If the cycle separator of r^* is a cut point, any eligible path passing it will break r^*. If the cycle separator, C, of r^* is non-trivial and an eligible path passes at least one edge of C, we can always adjust the eligible path to travel at least one half of C and delete the path. To finish the separation of r^*, now we only need to deal with the remain segment I of C, and I is at least one half shorter than C. We shorten I from its two ends until we get a segment I' so that I' is part of a simple cycle and still separates the current graph into connected components smaller than two third of the original r^*. Lemma 3 shows that this process always succeeds and uses $O(\log n)$ time and $n + m$ processors. We take the approach of constructing a cycle separator from a path separator [12] but use a careful parallel implementation.

Lemma 3 *For $n_0 \leq n$, if a simple path I separates a graph G of n_0 vertices into connected components no larger than $2n/3$, there is a segment of I which separates G into connected components smaller than $2n/3$ and is part of a simple cycle at the same time. Finding such a segment and such a cycle for the given I needs $O(\log n)$ time and $C(n,m)+n+m$ processors, where $C(n,m)$ is the number of processors required for finding connected components in $O(\log n)$ time on the CRCW PRAM.*

Proof: For G and I described in the lemma, we want to find a *key* segment, I', of I. I' still separates G into connected components no larger than $2n/3$. And if I' is one vertex shorter from either end, there will be a maximal connected component larger than $2n/3$ after I' is deleted. In this case, I' was proved to be part of a simple cycle in G [12]. We use the following method to find such a I'.

We order vertices on I so that $I = \{v_0, \cdots, v_l\}$. Suppose I separates G into k maximal connected components, M_i, where $0 \leq i \leq k-1$. After we find the size, w_i, of M_i for $0 \leq i \leq k-1$, we compute the lowest subscript, α_i, so that v_{α_i} on I shares at least an edge with M_i. For each vertex, v_j, on I, we calculate $\phi_j = \sum_{\alpha_i = j} w_i$. Then we can compute prefix sums $\Phi_i = \sum_{j=0}^{i} \phi_j$ and find an subscript α so that $\Phi_\alpha \leq 2n/3 \wedge \Phi_{\alpha+1} > 2n/3$. Obviously $I'' = \{v_\alpha, \cdots, v_l\}$ is almost what we want except that it may still be shorten from v_l. Using a process similar to that used to find the v_α from I, we can find a v_β from I'' so that $I' = \{v_\alpha, \cdots, v_\beta\}$. In the above whole process, except that finding maximal connected components uses $O(\log n)$ time and $C(n,m)$ processors [23, 10], all other calculations can be done in $O(\log n)$ time with $n+m$ processors using standard techniques [14]. ∎

Our purpose of building a cycle containing a key segment I' of I is that a new eligible path passing an edge of this new cycle can change its way through the cycle to pass at least one half of I'. Now we can apply what we have done to I to a remain segment of I', and so forth. Because each time we deal with a segment at least one half shorter than the previous one, we can break a R^*-vertex into components that are a constant factor smaller in $O(\log n)$ iterations.

We next show how to find an eligible path passing an edge of a given cycle if there is one. We add a vertex in the middle of any edge of the given cycle and claim that there is an eligible path passing at least an edge of the cycle if and only there is an eligible path passing the new vertex. Given a source s, a sink t, and a vertex r, we can check whether there is a simple path from s to t via r and actually find one if possible in $O(\log n)$ time with $n+m$ processors. Our method is adding an edge to link s and t so that there is a required simple path if and only if s, t, and r are in the same biconnected component of the new graph. The biconnected component problem can be solved in $O(\log n)$ time with $n+m$ processors [25].

We also need to study the case when no eligible path passes an edge of a cycle separator. Although we can not simply delete the separator because some vertices of the separator may still be on eligible paths, the following two lemmas show that the separator can be well simplified.

Lemma 4 *Given a cycle separator, C, of a R^*-vertex r^*, if no eligible MVDP path passes an edge of C, at most one vertex of C is on eligible paths.*

Proof: Suppose two distinct vertices, v_1 and v_2, of cycle C are on eligible paths, $P_1 = \{s_x, \cdots, v_1, \cdots, t_y\}$ and $P_2 = \{s_a, \cdots, v_2, \cdots, t_b\}$. Without lose of generality, P_1 joints C first at v_1 and P_2 joints C last at v_2. We can connect segments from P_1, C and P_2 into an eligible path, $P_3 = \{s_x, \cdots, v_1, \cdots, v_2, \cdots, t_b\}$, where $\{s_x, \cdots, v_1\}$ is part of P_1, $\{v_1, \cdots, v_2\}$ is part of C, and $\{v_2, \cdots, t_b\}$ is part of P_2. Because v_1 and v_2 are distinct, P_3 passes at least one edge of C. This contradicts the assumption of the lemma. ∎

If we look for MVDP and find no eligible path which passes an edge of a cycle separator of a R^*-vertex, all vertices on the cycle separator can be deleted but at most one. Such a remain vertex will be treated as a 'cut point' in the next round of simplification.

Lemma 5 *Given a cycle separator, C, of a R^*-vertex r^*, if no eligible S-T path passes an edge of C, no two vertices of C are in the same maximal connected component after all edges that are not on any eligible path are deleted from r^*.*

Proof: Since no eligible s-t path passes an edge of C, all edges between vertices of C are deleted when all edges not on any eligible path are deleted. Suppose a remain maximal connected component of r^* has two distinct vertices, v_1 and v_2, from C and both are on eligible paths. Without lose of generality, we assume that $P_0 = \{v_1, \cdots, v_2\}$ is a segment of C and no other vertex of P_0 is in r^*. Because v_1 and v_2 are in the same connected component, there must be a path $P_1 = \{v_1, v_x, \cdots, v_2\}$. Since (v_1, v_x) must be on an eligible path, there must be an eligible path $P_2 = \{s_i, \cdots, v_1, v_x, \cdots, t_i\}$. Then there is an eligible path $P_3 = \{s_i, \cdots, v_1\} \cup P_0 \cup \{v_2, \cdots, v_x, \cdots, t_i\}$, where P_3 starts from s_i, goes to v_1 via P_2, continues to v_2 via P_0, continues to v_x via P_1, and ends at t_i via P_2 again. This result contradicts our assumption that no eligible path passes an edge of C. ∎

If we look for S-T and find that no eligible path passes an edge of a cycle separator, C, of a R^*-vertex r^*, by Lemma 5, this R^*-vertex is divided into disconnected components each contains at most one vertex from C after all edges not on any eligible path are deleted. To finish the separation of r^*, remain vertices of C are treated as 'cut points' in the next round of simplification although the vertices may not be real cut points of their current components. When these 'cut points' are deleted in the succeeding process, r^* is broken into components that are a constant factor smaller.

Note that we did not mention how to delete all edges not on any eligible path in the proof of Lemma 5. Similar to our explanation of how to find an eligible path passing an edge of a cycle, this process is as easy as the biconnected component problem.

2.4 How to break R^*-vertices simultaneously with eligible paths

In order to break R^*-vertices simultaneously, we need a set of disjoint eligible paths so that the process used to break single R^*-vertex in the previous section can be applied independently to distinct R^*-vertices. We call a set of vertices and cycles in R^*-vertices as *key vertices* and *key cycles* if each R^*-vertex has at most one key vertex or one key cycle but not both. Every key cycle has a segment called *key segment.* Initially we let key vertices and key cycles be cut points and cycle separators in R^*-vertices, and a key segment of a cycle separator is formed by removing one edge from the cycle. Given key vertices and key cycles, we call a set of disjoint eligible paths as

a *cut set* if (1) each path passes either a key vertex or an edge of a key cycle; (2) no two paths joint the same key point or key cycle; (3) the set is maximal. By the definition of key point and key cycle and the definition of cut set, we can perform the following process that uses cut sets to consume key points and key segments piece by piece on R^*-vertices in parallel.

Given key points, key cycles, and a corresponding cut set, (1) delete all paths in the cut set which pass a key point; (2) reroute rest paths in the cut set so that each overlaps at least one half of a key segment via a key cycle it passes, and then delete new paths; (3) delete all key points not on a path of the cut set; (4) if a key cycle does not joint a path in the cut set and S-T is our target, delete all edges in the corresponding R^*-vertex which are not on any eligible path after the first three steps; (5) if a key cycle does not joint a path in the cut set and MVDP is our target, delete all vertices on the key cycle which are not on any eligible path.

Lemma 6 *The above process correctly deletes a set of disjoint eligible paths and some redundant vertices and edges. After the process, the maximum length of remain key segments is at least one half shorter than that before the process and all remain key segments are in distinct R^*-vertices. It uses $O(\log n)$ time and $n + m$ processors.*

Proof: Since the referred process is executed in distinct R^*-vertices in parallel, new eligible paths formed in Step (2) are disjoint. Therefore the process only deletes disjoint eligible paths in Step (1) and Step (2). A key vertex not on a path of the cut set is not on any additional eligible path because the cut set is maximal. So Step (3) only deletes some redundant vertices. It is self evident that Step (4) and Step (5) only delete redundant vertices and edges.

All key points are deleted in Step (1) and Step (3). If a key segment is in a key cycle jointed by an eligible path in the cut set, at least half of it will be deleted in Step (2) and its remain is unbroken. All other key segments will be break into vertices in distinct R^*-vertices in Step (4) and Step (5) by Lemma 5 and by Lemma 4. This proofs that the remain key segments are one half shorter and are in distinct R^*-vertices.

The computation complexity of this process directly follows the proofs of previous lemmas. ∎

The above process produces a new graph in which all remain key segments are in distinct R^*-vertices by Lemma 6. Therefore we can construct cycles each contains a useful part of a remain key segment by Lemma 3 simultaneously. Obviously we can treat newly constructed cycles as key points and key cycles and let new key segments be overlaps of current key cycles and previous key segments. The longest new key segment is at least one half shorter than the previous. Given necessary cut sets, we can totally delete initial key segments in $O(\log n)$ iterations of the process described so far and break initial R^*-vertices into components that are a constant factor smaller.

Before concluding our algorithms, we need to present how to find a cut set for given key points and key cycles. Finding a cut set is similar to finding maximal vertex-disjoint paths on a tripartite graph except that this time an 'eligible' path has to meet more restrictions such as it has to pass an edge of a key cycle.

Lemma 7 *Given a graph G and its key vertices and key cycles, finding a cut set for a S-T algorithm uses $O(\log^3 n)$ time and $n+m$ processors and finding a cut set for a MVDP algorithm uses $O(\log^4 n)$ time and n^5 processors.*

Proof: For the S-T problem, we construct a tripartite graph G' in which R-vertices represent key points and key cycles, and each pair of matched source and sink are connected to a key point or a key cycle if and only if at least one eligible path from the source to the sink passes the key point or an edge of the key cycle accordingly. Maximal s-t paths of G' imply a cut set of G. Finding G' is as easy as finding biconnected components.

For the MVDP problem, we construct a graph G' in which a vertex represents a triple that consist of a source, a sink, and either a key point or a key cycle. A triple is in G' if and only if there is at least one eligible path from the source to the sink via the key point or an edge of the key cycle accordingly. Two vertices of G' share an edge if and only if the two corresponding paths joint the same source, the same sink, or either the same key point or the same key cycle. The most complicate G' has no more vertices than G. Obviously a MVDP of G' implies a cut set of G. The rest is only a repeat of the proof of Lemma 2. ∎

2.5 Algorithms and Analysis

Based on the techniques introduced so far, we now complete Algorithm 1 with details.

The First Step of Algorithm 1 (Details)

While the current graph is not tripartite **do** {

Find a cycle separator in every R^*-vertex as an initial key cycle and denote the size of the largest R^*-vertex as n_0;

While any R^*-vertex is still larger than $\frac{2n_0}{3}$ **do** {

1. Find a cut set for the current key vertices and key cycles as described in Lemma 7;
2. Break the current graph with the cut set as described in Lemma 6 and put all deleted eligible paths in PATHS;
3. Find cycles containing useful parts of remain key segments to form new key points and key cycle as described in Lemma 3. }}

Lemma 8 *The first step of Algorithm 1 uses $O(\log n(T_{sep}(n) + \log^4 n))$ time and $P_{sep}(n) + n + m$ processors for the S-T problem, and uses $O(\log n(T_{sep}(n) + \log^5 n))$ time and $P_{sep}(n)+n^5$ processors for the MVDP problem.*

Proof: The outside *while* loop has $O(\log n)$ iterations because the maximal R^*-vertex is one half smaller each time. Each outside *while* loop finds cycle separators and runs the inside *while* loop $O(\log n)$ iterations because the maximal key segment is one half shorter each time. The rest follow Lemma 7. ∎

Theorem 1 *Algorithm 1 finds maximal s-t paths in $O(\log n(T_{sep}(n) + \log^4 n))$ time using $P_{sep}(n)+ n + m$ processors and finds maximal vertex-disjoint paths in $O(\log n(T_{sep}(n) + \log^5 n))$ time using $P_{sep}(n) + n^5$ processors.*

Proof: The algorithm's correctness is based on an invariant of the *while* loops in its first step. The invariant is that the union of paths in PATHS and maximal vertex-disjoint paths in the remain graph is maximal vertex-disjoint paths in the original graph. The algorithm's complexity follows Lemma 8, Lemma 1, and Lemma 2. ∎

Algorithm 1 is DNC if finding cycle separators in R^*-vertices is in DNC, and there are DNC cycle separator algorithms for planar graphs [22]. So we have the following result.

Corollary 1 *There is a DNC algorithm that finds maximal s-t paths and maximal vertex-disjoint paths in any undirected graph whose maximal connected components of intermediate vertices are planar graphs.*

3 Find vertex-disjoint paths in undirected planar graphs

In this section, we discuss a linear-processor DNC MVDP algorithm for undirected planar graphs in which cycle separators can be found in $O(\log n)$ time by n processors [22]. The algorithm also depends on efficient parallel algorithms that find maximal matching [19, 7] and maximal connected components [23, 10] in a planar graph in $O(\log n)$ time on n processors.

For the S-T problem, Algorithm 1 can maintain planarity. So we simply use $O(\log n)$ time linear-processor algorithms [7, 23, 10, 22] for planar graphs to replace those for general graphs.

Theorem 2 *Maximal s-t paths in an undirected planar graph can be found in $O(\log^3 n)$ time with n processors.*

In contrast with the S-T problem, the MVDP problem has no straight forward linear-processor algorithm for planar graphs because planarity can be seriously damaged in Algorithm 1. For example, original methods for finding MVDP in a tripartite graph and finding a cut set were not designed for planar graphs. By exploring planarity, we can find MVDP in a tripartite planar graph and priority maximal matching in a bipartite planar graph in $O(\log^2 n)$ time on n processors. Based on the two new algorithms, we find and use a new type of cut sets to simplify maximal connected components of intermediate vertices so that we can retain the over all approach of Algorithm 1.

3.1 A DNC MVDP algorithm for tripartite planar graphs

For the purpose of finding MVDP in a tripartite graph, isolated sources, isolated sinks, and R-vertices not linked to both a source and a sink are redundant. Removing redundant vertices from a tripartite planar graph, referred as *remove redundance*, only needs $O(\log n)$ time and n processor.

Algorithm 2 *Find-MVDP-in-Tripartite-Graphs*

Input: A tripartite graph, $G = (S \cup R \cup T, E)$
Output: A set, PATHS, containing MVDP of G
While the current G is not empty **do**

1. Remove redundance, find a maximal matching between sources and R-vertices, and then find a maximal matching between matched R-vertices and sinks. If a R-vertex r is first matched to a source s and then matched to a sink t, delete path $\{s, r, t\}$ from G and add this path to PATHS. Delete from G all R-vertices that were matched to sources but not to sinks.

2. Repeat Step 1 on the current G, but change roles of sources and sinks and switch positions of symbol s and t.

3. Construct a new graph, $G' = (S' \cup R' \cup T', E')$, from the current G as the following: let every R-vertex retains only one of its edges shared by sources and then merge R-vertices linked to the same source into one vertex in R'. Find a maximal matching between R' and sinks. If r' in R' is matched to s in S' and to t in T', select a R-vertex r merged inside r' which is connected to t in G, add path $\{s, r, t\}$ to PATHS, and delete this path from G. Otherwise, delete from G all R-vertices inside those r' in R' which were not matched to sinks.

Lemma 9 *Let G denote a maximal connected component at the beginning of an iteration of Algorithm 2 and G' denote the remain graph of G at the end of the iteration, there are several useful facts: (1) the union of vertex-disjoint paths found from G in the iteration and MVDP to be found in G' is MVDP of G; (2) at least as many edges as remain vertices in G' are deleted from G in the iteration; (3) each iteration can be done in $O(\log n)$ time with n processors when G is a tripartite planar graph.*

Proof: The proof of fact (1) is trivial. This fact guarantees that Algorithm 2 returns correct MVDP when it stops. The fact (2) is proved by the following analysis. Every remain source loses at least one edge in Step 1 because all R-vertices in a maximal matching between sources and R-vertices are deleted one way or another. For the same reason, every remain sink loses at least one edge in Step 2. Every remain R-vertex loses at least one edge in Step 3 because it loses an edge to a source deleted with an eligible path. Given a tripartite planar graph, non-trivial computations in Algorithm 2 are removing redundance and finding a maximal matching of a bipartite planar graph. The processes can easily be done in $O(\log n)$ time with $O(n)$ processors as indicated before. ∎

Lemma 9(2) is significant for a tripartite planar graph which is sparse in terms of edge number.

Theorem 3 *Given a tripartite planar graph of n vertices, three iterations of the* while *loop of Algorithm 2 either leave less than $n/2$ vertices or reduce the size of the largest connected component by half. So Algorithm 2 finds MVDP in a tripartite planar graph in $O(\log^2 n)$ time using n processors.*

Proof: Obviously $m \leq 2n - 4$ in a tripartite planar graph. Without lose of generality, we assume that G is connected. If less than $n/2$ vertices are left after three iterations, the lemma is true. Otherwise at least $3n/2$ edges are deleted because $n/2$ remain vertices imply that at least $n/2$ edges are deleted in each iteration by Lemma 9(2). So no remain connected component is larger than $n/2 - 5$ because at most $n/2 - 4$ original edges are left. In fact, the decomposition of a

tripartite planar graph is faster than this analysis shows because the analysis doesn't count in edges deleted with vertex- disjoint paths and redundant vertices. Therefore G will be empty after $O(\log n)$ iterations. Because each iteration uses $O(\log n)$ time and n processors according to Lemma 9, Algorithm 2 uses $O(\log^2 n)$ time and n processors in total. ∎

3.2 Find priority maximal matching in a priority bipartite planar graph

A priority bipartite graph is a bipartite graph that has positive weights on its edges. A priority maximal matching of a priority bipartite graph is a maximal matching whose total weight can not be increased by changing one source's matcher to a sink not matched. It is easy to see that the priority maximal matching problem is simpler than the maximum matching problem but is more complicate than the maximal matching problem. In this paper a priority maximal matching algorithm is used in the following situation. Suppose there is a graph in which R^*-vertices are simple paths and every R^*-vertex is linked to exactly one distinct source from one end, we want to find a set of vertex-disjoint paths so that no R-vertex is linked to a sink after the paths are deleted. Clearly this problem can be reduced to the priority maximal matching problem. Later we will see that the paths in the question are key segments and a priority maximal matching algorithm is used to find a desired cut set. Using $O(\log n)$ time and n processors, we can simplify a priority bipartite planar graph so that: (1) if several dangling edges incident to the same vertex, only the largest edge is retained; (2) if two edges incident to the same vertex whose degree is 2 are called a pseudo edge and several pseudo edges share the same end points, only two pseudo edges are retained, where the first pseudo edge contains the largest edge incident to one end point among all edges in the pseudo edges and the second pseudo edge contains the largest edge incident to the other end point among edges in the rest pseudo edges. A priority maximal matching in a simplified priority bipartite graph is the same in the original graph. We observed that there are roughly the same number of useful sources and useful sinks in a bipartite planar graph.

Lemma 10 *If $G = (S \cup T, E)$ is a simplified priority bipartite planar graph, $n_s/9 < n_t < 9n_s$.*

Proof: Since G is simplified, $|\{t \mid \Delta(t) = 1 \wedge t \in T\}| \leq n_s$. Consider a subgraph, $G' = (S' \cup T', E')$, of G, where T' only has sinks whose degrees are 2 and S' only has sources which are adjacent to sinks in T'. G' is still a priority bipartite planar graph. If we treat two edges incident to the same sink of G' as one edge, the new graph has no more than two parallel edges between any two vertices because G is simplified. Obviously $|T'|$ is no larger than two times of the maximum edge number of a planar graph that has $|S'| \leq n_s$ vertices. So $|\{t \mid \Delta(t) = 2 \wedge t \in T\}| \leq 2(3n_s - 6)$. Because $|\{t \mid \Delta(t) \geq 3 \wedge t \in T\}|$ is at most $m/3$ and $m \leq 2(n_t + n_s) - 4$, $3|\{t \mid \Delta(t) \geq 3\}| < 2n_t + 2n_s - 4$. Now $|\{t \mid \Delta(t) \geq 3\}| < n_t$ implies $|\{t \mid \Delta(t) \geq 3\}| < 2n_s - 4$. Therefore $n_t = |\{t \mid \Delta(t) = 1 \vee \Delta(t) = 2 \vee \Delta(t) \geq 3\}| < n_s + 6n_s - 12 + 2n_s - 4 < 9n_s$. The above inference is still correct when the roles of sources and sinks are switched, which leads to $n_s/9 < n_t < 9n_s$. ∎

Algorithm 3 *Find-Priority-Maximal-Matching*

Input: A priority bipartite planar graph $G = (S \cup T, E)$
Output: A priority maximal matching, PMM, of G
While the current G is not empty **do** {

Simplify G, construct a subgraph G' in which every source of G retains the largest edge incident to it, find a maximal matching in G', add this maximal matching to PMM, delete matched vertices and remain edges of G' from G. }

Lemma 11 *Suppose the largest maximal connected component of a bipartite planar graph, $G = (S \cup T, E)$, has n_0 vertices, after Algorithm 3 runs* 30 *iterations of its* while *loop, the remain graph either has less than $n_s/2$ sources or has no maximal connected component larger than $n_0/2$.*

Proof: Without lose of generality, we assume that G is connected and simplified. Since G has less than $9n_s$ sinks by Lemma 10, G has less than $10n_s$ vertices and less than $20n_s - 4$ edges in total by Euler's Theorem. If there are still $n_s/2$ sources left after the 30th iteration of Algorithm 3, at least $15n_s$ edges has been deleted because each remain source has an edge deleted in each iteration. In this case, at most $5n_s - 4$ edges are left, and any remain connected component has at most $5n_s - 3$ vertices which are less than $n/2$. ∎

Theorem 4 *There is an algorithm that finds a priority maximal matching in a priority bipartite planar graph in $O(\log^2 n)$ time using n processors.*

Proof: The algorithm is correct because concatenate maximal matchings are all based on edges having the highest priority at the time of execution and only sure redundant edges are deleted. By Lemma 11, any priority bipartite planar graph of n vertices will be reduced to empty within $O(\log n)$ iterations of the *while* loop of Algorithm 3. Each iteration uses $O(\log n)$ time and n processors. ∎

It should be pointed out that finding DNC priority maximal matching algorithms for general bipartite graphs is still an open problem and is interesting of its own.

3.3 A DNC MVDP algorithm for undirected planar graphs

For the MVDP problem on planar graphs, the original algorithm for finding a cut set in Lemma 7 may damage planarity thus use more than n processors. So we use a different kind of 'cut set', called *perfect-cut* in this paper, to reduce sizes of R^*-vertices. Before defining perfect-cut, we introduce a few obvious but useful facts without proofs. If G is a planar graph, so is G^*. Any MVDP in G^* can be extended into disjoint eligible paths in G. Given any MVDP in G^*, if a R^*-vertex is not on the MVDP, none of its vertices will be on an eligible path after disjoint eligible paths extended from the MVDP are deleted from G.

Given G and cycle separators in its R^*-vertices, we want to find a MVDP in G^* which can be extended into a particular set of eligible paths in G. If a R^*-vertex r^* is on the MVDP, after corresponding eligible paths are deleted from G, vertices on the cycle separator of r^* either will have been deleted with the eligible paths or will be no longer on any eligible path. Given such

a MVDP in G^*, a R^*-vertex either becomes redundant because it is not on the MVDP, or is broken into R^*-vertices of a constant factor smaller by an eligible path because its cycle separator can be removed one way or another as just described. It is this kind of MVDP in G^* and their corresponding eligible paths in G that are called perfect-cut.

We start to explain how to find a perfect-cut from the following simple case. Given some R^*-vertices in G which are one-to-one matched to sources, how to decompose the R^*-vertices efficiently and simultaneously when they share sinks? Since these R^*-vertices are matched to distinct sources, we can find, in parallel, path separators in the R^*-vertices which are rooted at corresponding sources. For each R^*-vertex r^* that has a path separator rooted at source s, we give edge (r^*, t) a weight which is equal to the length of the maximum overlap of the path separator in r^* and an eligible path from s to t via vertices in r^*. Therefore these R^*-vertices and adjacent sinks form a priority bipartite graph which is planar when G is planar. A priority maximal matching in this priority bipartite graph can be extended into a set of disjoint eligible paths in G. If a R^*-vertex is in the priority maximal matching, a segment of its path separator can be deleted as part of an eligible path extended from the matching, and the rest vertices on its path separator are no longer on any eligible path after all eligible paths extended from the matching are deleted. This can be verified by the following explanation. Because all remain sinks are not in the priority maximal matching, if a remain vertex of a path separator of a R^*-vertex in the matching is on a path to a remain sink, it will contradict the fact that the matching is a priority maximal matching as described.

We extend the idea of using priority maximal matching to general cases. Since a perfect-cut is a MVDP in the tripartite planar graph G^*, finding a perfect-cut is similar to finding MVDP in a tripartite graph. In Algorithm 2, when R-vertices are matched to sources (sinks), a maximal match between matched R- vertices and sinks (sources) is used to decide eligible paths. For the purpose of finding a perfect-cut, when R^*-vertices are matched to sources (sinks), we have to use the process described in the previous paragraph to form a priority bipartite graph between matched R^*-vertices and sinks (source) so that a priority maximal matching can be extended to a set of eligible paths which can remove cycle separators in these R^*-vertices in one round.

Algorithm 4 *Find-A-Perfect-Cut*

Input: A graph, $G = (S \cup R \cup T, E)$
Output: A perfect-cut of G
Mark all R^*-vertices of G^* as active and find a cycle separator in every of them;
While G^* still has active R^*-vertices **do** {

1. Remove redundance, find a maximal matching between sources and active R^*-vertices, extend edges in the matching into path separators based on cycle separators, and then find a priority maximal matching between matched R^*-vertices and sinks according to the path separators as described before. If a R^*-vertex r^* is matched to a source s is also matched to a sink t, delete a corresponding eligible path, $\{s, \cdots, t\}$, from G, delete all remain vertices on the path

separator of r^* from G, and mark all new R^*-vertices formed by decomposed r^* as inactive. Delete R^*-vertices that are matched to sources but not to sinks from G^* and G.

2. Repeat Step 1 on the current G^*, but change roles of sources and sinks and switch positions of symbol s and t.

3. Let every active R^*-vertex retains only one of its edges shared with sources, extend the edge into a path separator via the cycle separator in the R^*-vertex, construct a priority bipartite graph between active R^*-vertices and sinks, merge active R^*-vertices linked to the same source into one *conglomerate vertex*, keep the largest among parallel edges, and find a priority matching between conglomerate vertices and sinks. If a conglomerate vertex is matched to s and to t, select a R^*-vertex r^* inside the conglomerate vertex which possesses the edge that link the conglomerate vertex to t in the priority matching, deleted an eligible path $\{s, \cdots, t\}$ from G, and mark all remain parts of r^* as inactive. Delete from G all R^*-vertices inside those conglomerate vertices not matched to sinks. }

Lemma 12 *There is an algorithm that finds a perfect-cut in an undirected planar graph in $O(\log^3 n)$ time using n processors.*

Proof: The complexity analysis is similar to that of Theorem 3 except that our $O(\log^2 n)$ time priority maximal matching algorithm replaces a $O(\log n)$ time conventional maximal matching algorithm in some places of Algorithm 2. ∎

If we use the parallel algorithm that finds perfect-cut to replace the inside *while* loop in the first step of Algorithm 1 and use Algorithm 2 to replace the second step of Algorithm 1, we get a linear-processor MVDP algorithm for undirected planar graphs.

Theorem 5 *Maximal vertex-disjoint paths in an undirected planar graph can be found in $O(\log^4 n)$ time with n processors.*

There should be faster MVDP algorithms for planar graphs which make a better use of planarity. But it also will be harder to extend them to general graphs.

Remarks. For undirected graphs, we have reduced a DNC MVDP vertex-disjoint paths algorithm to a DNC DFS algorithm. For undirected planar graphs, we have presented linear-processor DNC algorithms for the S-T problem and the MVDP problem. However, it is still unknown whether our reduction is 'strict' under the meaning that a DNC maximal vertex-disjoint paths algorithm for undirected graphs exists and is independent of whether DFS is in DNC. On the other hand, it is also interesting enough to know whether the DFS problem can be reduced to the MVDP problem and whether the generalized MVDP problem can be reduced to the DFS problem. Their answers will lead to a better understanding of problems related to vertex-disjoint paths and depth-first search in parallel computation.

Acknowledgement. I am grateful to Gregory Shannon for his valuable comments, reading the drafts of this paper, and encouragements.

References

[1] A. Aggarwal and R. Anderson. A random NC algorithm for depth first search. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 325–334, 1987.

[2] A. Aggarwal, R. Anderson, and M. Kao. Parallel depth-first search in general directed graphs. In *Proceedings of the 21th Annual ACM Symposium on Theory of Computing*, pages 297–308, 1989.

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

[4] R. Anderson. A parallel algorithm for the maximal path problem. *Combinatorica*, 7(3), 1987.

[5] R. Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4), August 1988.

[6] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *J. Theoret. Comput. sci.*, 10:111–121, 1980.

[7] A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434–446, November 1988.

[8] A. Goldberg, S. Plotkin, and P. Vaidya. Sublinear-time parallel algorithms for matching and related problems. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 174–185, 1988.

[9] M. Goldberg and T. Spencer. A new parallel algorithm for the maximal independent set problem. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 161– –165, 1987.

[10] T. Hagerup. *Optimal Parallel Algorithms on Planar Graphs.* Technical Report 1987-16, Informatik, Universitaet des Saarlandes, Saarbruecken, West Germany, October 1987.

[11] A. Israli and Y. Shiloach. An improved parallel algorithm for maximal matching. *Information Processing Letters*, 22:57– –60, January 1986.

[12] M. Kao. All graphs have cycle separators and planar directed depth-first search is in DNC. In *Proceedings of the 3rd Agean Workshop on Computing: VLSI Algorithms and Architectures*, pages 53–63, 1988.

[13] M. Kao and P. Klein. Toward overcoming the transitive-closure bottleneck: efficient parallel algorithms for planar digraphs. In *Proceedings of the 22th Annual ACM Symposium on Theory of Computing*, pages 181–192, 1990.

[14] R. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. In the *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., North-Holland, Amsterdam 1990.

[15] R. M. Karp. On the complexity of combinatorial problems. *Networks*, 5:45–68, 1976.

[16] S. Khuller, S. G. Mitchell, and V. V. Vazirani. Processor efficient parallel algorithms for the two disjoint paths and for finding a kuratowski homeomorph. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 300–305, 1989.

[17] S. Khuller and B. Schieber. Efficient parallel algorithms for testing connectivity and find disjoint s-t paths in graphs. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 288–293, 1989.

[18] J. F. Lynch. The equivalence of theorem proving and the interconnection problem. *ACM SIGDA Newsletter*, 5(3), 1976.

[19] S. Micali and V. Vazirani. An $O(|E|\sqrt{|V|})$ algorithm for finding maximal matchings in general graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, pages 17–27, 1980.

[20] J. Reif. Depth-first search is inherently sequential. *Information Processing Letters*, 20:229–234, June 1985.

[21] N. Robertson and P. D. Seymour. Disjoint paths – a survey. *SIAM J. Alg. Disc. Meth.*, 6(2):300–305, April 1985.

[22] G. Shannon. A linear processor algorithm for depth-first search in planar graphs. *Information Processing Letters*, 29(3):119–124, October 1988.

[23] Y. Shiloach and U. Vishkin. An $O(\log n)$ parallel connectivity algorithm. *Journal of Algorithms*, 3:57–67, 1982.

[24] J. Smith. Parallel algorithms for depth-first searches I. planar graphs. *SIAM Journal on Computing*, 15(3):814–830, August 1986.

[25] R. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal on Computing*, 14(4):862–874, November 1985.