

Parallel algorithm for shortest pairs of edge-disjoint paths

S. Banerjee, R. K. Ghosh[†] and A.P.K. Reddy
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur 208 016, INDIA

Abstract

Let $G = (V, E)$ be a directed graph having a non-negative cost associated with each edge. Let $s \in V$ be a special vertex called the source and $W \subset V$ be a set of other vertices called sinks in G . In this paper, a parallel algorithm is proposed for finding a pair of edge-disjoint paths from s to each possible sink $t \in W$ such that the sum of the costs of the two paths is minimized. This algorithm has processor and time complexities same as those needed to find shortest paths from s to all sinks $t \in W$, i.e., $n^3/\log n$ processors and $O(\log^2 n)$ time.

1 Introduction

Let $G = (V, E)$ be a directed graph having a non-negative cost $c(u, v)$ associated with each edge $(u, v) \in E$ and let $|V| = n$ and $|E| = m$. Let $s \in V$ be a special vertex called the source and $W \subset V$ be a set of other vertices called sinks in G . In this paper, we consider the problem of finding in parallel a minimum-weight pair of edge-disjoint paths from s to each sink $t \in W$. This problem, hereinafter, is referred to as the multiple-sink problem (*MSP*). When only one sink is specified we have the single-sink i.e., $|W| = 1$ problem (*SSP*).

A sequential algorithm for solving *SSP* requires two iterations of Dijkstra's shortest path algorithm [Dij 59] as indicated in section 2 of this

[†]Email: rkg@iitk.ernet.in

paper. Therefore, if the upper bound of an efficient implementation of Dijkstra's algorithm is denote by $S(n, m)$, SSP can also be solved in $O(S(n, m))$ time sequentially . This implies a trivial solution for MSP with n sinks can be obtained by $2n$ applications of Dijkstra's algorithm, i.e., in $O(nS(n, m))$ time. However, Tarjan and Suurballe [ST 84] have shown that MSP can be solved in $O(m \log_{1+m/n} n)$ time. More precisely, they proposed an $O(S(n, m))$ time algorithm for MSP . Since the best available value for $S(n, m)$ at the time was $O(m \log_{1+m/n} n)$ Tarjan and Suurballe quoted this expression for time bound. Currently, the best available value for $S(n, m)$ is $O(\min\{m + n \log n, m \log \log C, m + \sqrt{n \log C}\})$ [FT 84, John 82, AMOT 90] where C represents the largest edge cost in G . Hence, following Tarjan and Suurballe's method a solution for MSP can be found within the same time bound.

The problem of finding all pairs of shortest paths in a directed graph with non-negative edge weights can be solved in $O(\log^2 n)$ time using $n^3/\log n$ processors on a CREW PRAM [GB 86]. Therefore, SSP can be solved within the same resource bounds. However, no efficient parallel algorithm has yet been reported for MSP in parallel. In this paper we propose an algorithm on CREW PRAM for MSP which runs in $O(\log^2 n)$ time and requires $n^3/\log n$ processors, i.e., the bounds are same as those for solving for SSP . The bottleneck of the above algorithm is the processor and time complexity needed for computing single source shortest path. All the other steps in this algorithm use only $n^2/\log n$ processors and run in $O(\log n)$ time.

The paper is organized as follows. Section 2 provides a quick sketch of the sequential algorithm for SSP . Section 3 describes the parallel algorithm for MSP . The theoretical background for correctness of the algorithm is discussed in section 4. Section 5 deals with the details of the implementation of the algorithm within the time and processor bounds as that required for computing all pairs shortest paths.

2 Single sink algorithm

All paths in this paper refer to directed paths unless stated otherwise. Similarly, disjoint paths refer to edge-disjoint paths. A path from vertex v_1 to v_2 is denoted by $p(v_1, v_2)$.

The problem of finding a shortest pair of disjoint paths from s to a single sink v can be formulated as a the minimum-cost flow problem. In this version of the min-cost flow problem all arc capacities are unity and a total flow of value 2 must be sent from s to v such that the cost of the flow is minimized. This version of the flow problem can be solved by two successive iterations of the shortest path algorithm [AMO 93]. For the sake of completeness a summary of the proof of the formulation of *SSP* as the flow problem is given in the appendix to this paper.

A sketch of the algorithm for solving *SSP* in terms of a min-cost flow problem is presented here, as this is crucial to the understanding of the parallel algorithm for *MSP* presented later in the next section.

Step 1 [Computation of shortest paths].

Obtain the shortest path tree T rooted at s .

Comment: Let the cost of the shortest path from s to each vertex $x \in G$ be denoted by $C(s, x)$. Let the shortest path from s to v be denoted as P_1 .

Step 2 [Reduced cost transformation].

The cost of every edge (a, b) in G is transformed to $c'(a, b) = c(a, b) + C(s, a) - C(s, b)$.

Comment: All the edges belonging to T will have a reduced cost equal to 0 due to this transformation.

Step 3 [Modification of graph].

Reverse the orientation of all edges of G which lie in P_1 to form a new graph G_v .

Comment: Hereinafter, reversed edges refer to these re-oriented edges. Also,

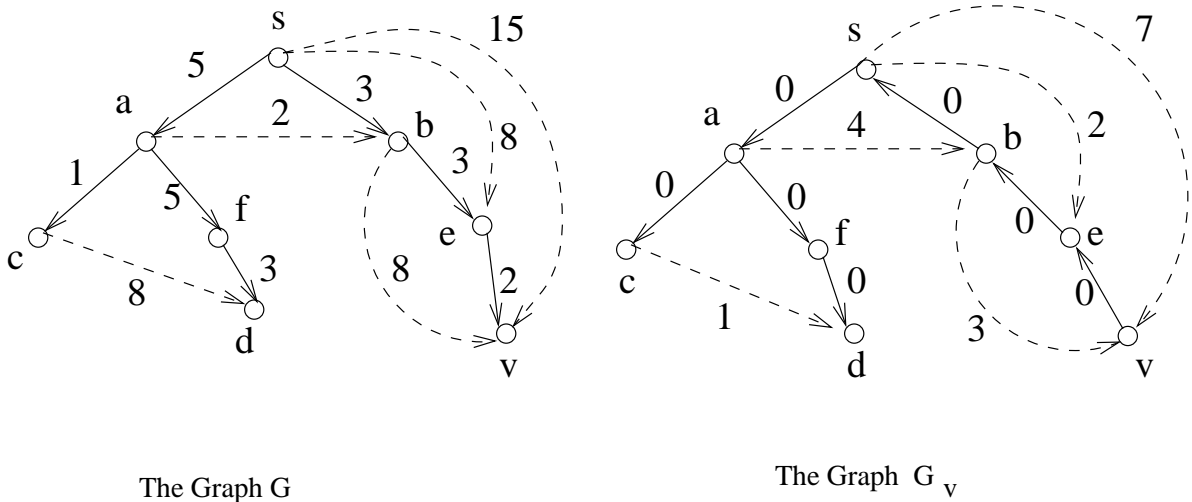


Figure 1: (a) A weighted graph G (b) Graph G_v

tree edges in G_v refer to all the edges in T , including the reversed edges.

Step 4 [Computation of shortest path].

Compute the shortest path from s to v in G_v . Let it be denoted as P_2 .

Step 5 [Path pair Generation].

The desired path pair, say P'_v and P''_v is obtained by deleting all edges belonging to $P_1 \cap P_2$ from the set of edges $P_1 \cup P_2$. An example is illustrated by Figure 1(e).

3 Multiple sink parallel algorithm

A simple way to obtain a multiple-sink algorithm would be to compute G_v for each sink v , as described in section 2, and perform the single-sink computation independently on each G_v . However, we represent the relevant information present in G_v for all vertices v in the form of an auxiliary graph G' . Conceptually, since tree edges do not contribute to the cost of a path in G_v , only the information about the nontree edges need to be maintained in

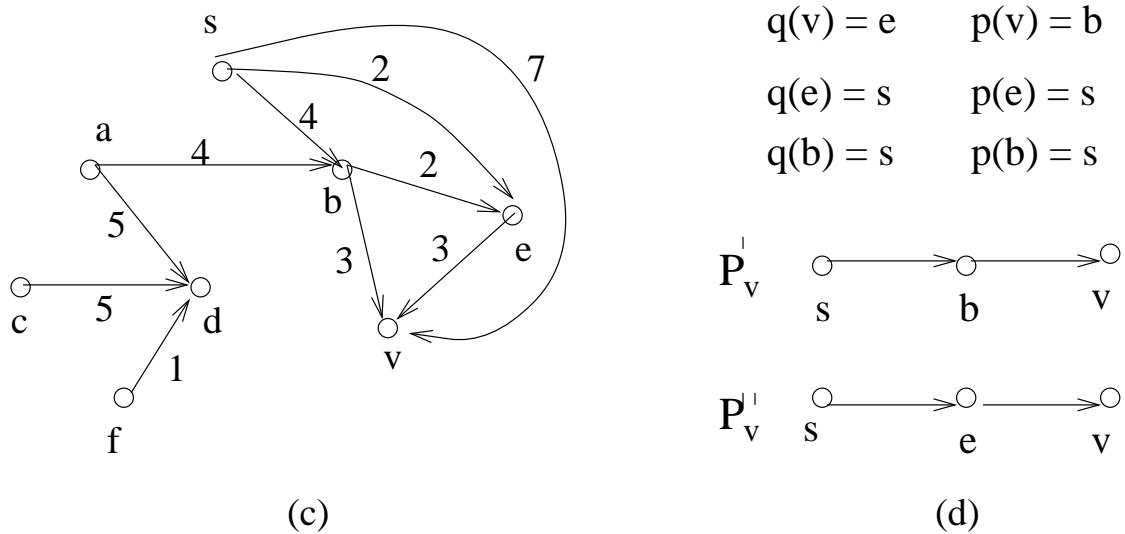


Figure 1: (c) Auxiliary graph G' (d) Path construction

some form.

The major problem, therefore, is to construct an auxiliary graph of the required kind. In this section we describe the basic steps of the parallel algorithm for multiple sink problem. The correctness of constructing G' with required properties is discussed in the next section.

Step 1 [Construction of shortest path tree for G].

Compute the shortest path tree T rooted at s .

Comment: In Figure 1(a) we have a graph G with the edges belonging to T marked by unbroken lines.

Step 2 [Reduced cost transformation].

Reduce the cost on all edges as defined in Step 2 of the single-sink algorithm.

Step 3 [Auxiliary graph construction].

Construct a new graph $G' = (V', E')$ as follows:

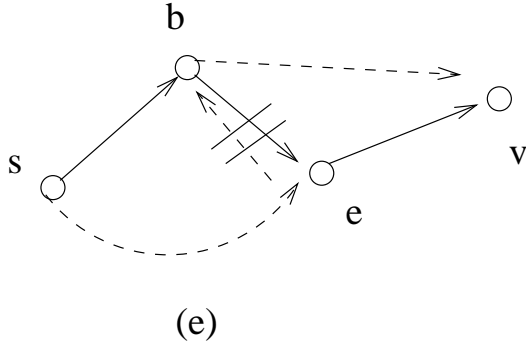


Figure 1: (e) Deletion of edges common to P_1 and P_2

Step 3.1 Set $V = V'$ and $E' = \phi$.

Step 3.2 For each nontree edge $(a, b) \in E - T$ do

Let V^* denote the vertices lying in the undirected path in T given by $p(LCA(a, b), a) \cup p(LCA(a, b), parent(b))$, where $parent(b)$ denotes the parent of b in T .

For each $v' \in V^*$ do Set $E' = E' \cup \{(v', b)\}$ and $c(v', b) = c(a, b)$.

Comment: Here $LCA(a, b)$ denotes the lowest common ancestor of $(a, b) \in T$. Figure 1 illustrates an example for the construction of G' . It should be pointed out that following the edge-construction method faithfully would potentially result in insertion of multiple edges from v' to b . For e.g., there will be two edges from e to v , one with cost 7, the other with cost 3. In such a case, only a single edge should be added in G' , assigning the value of the minimum cost edge to it.

Step 4 [Construction of shortest path tree for G'].

In G' , compute the shortest path tree T' rooted at s . For each vertex x define $q(x)$ as predecessor of x in T' and $p(x)$ as tail of the nontree edge in G which caused the addition of the edge $(q(x), x)$ in G' .

Step 5 [Generation of pair of paths].

For each vertex v construct the shortest path pair, P'_v and P''_v , in the following manner :

Mark all the vertices in the shortest path from v to s in G .

Generate the paths by two iterations of the step *traversal* given below. In each iteration one of the desired paths is generated.

Traversal: Define $x = v$, the path to be empty and repeat the following steps until $x = s$. If x is marked then unmark it and add $(p(x), x)$ to the path, else add (y, x) to the path where y is the parent of x in the shortest path tree T .

4 Correctness of the algorithm

We claim that G' , by construction, contains necessary information such that given the shortest path to any v in G' we can compute the shortest path for that v in its corresponding auxiliary graph G_v .

Consider a path, if it exists, from s to v in G_v . Such a path must be a sequence of one or more nontree edges (section 2, Step 3) with zero or more tree edges (possibly reversed) between two successive nontree edges. Since the cost of a tree edge is reduced to zero, the cost of this path depends only on that of the nontree edges.

Now consider the relationship between two successive nontree edges (a,b) and (x,y) in G_v in the form of claims given below (here, the successive nontree edges mean that there is a directed path from b to x consisting of zero or more tree edges).

Claim I. If there exists a path $p(v_1, v_2)$ in G_v , then there is either a) a path $p'(v_1, v_2)$ in G' of the same weight or b) the path $p(v_1, v_2)$ can't belong to any s -to- v shortest path in G_v .

Claim II. If there exists an edge (a,b) in G' such that there is no corresponding path of the same weight in G_v then the edge (a,b) can't belong to any s -to- v shortest path in G' .

Before proceeding to prove the claims, it is instructive to have a closer look at Figure 1(c) from the perspective of these claims.

First consider claim I. Since s does not belong to the tree path $c -$

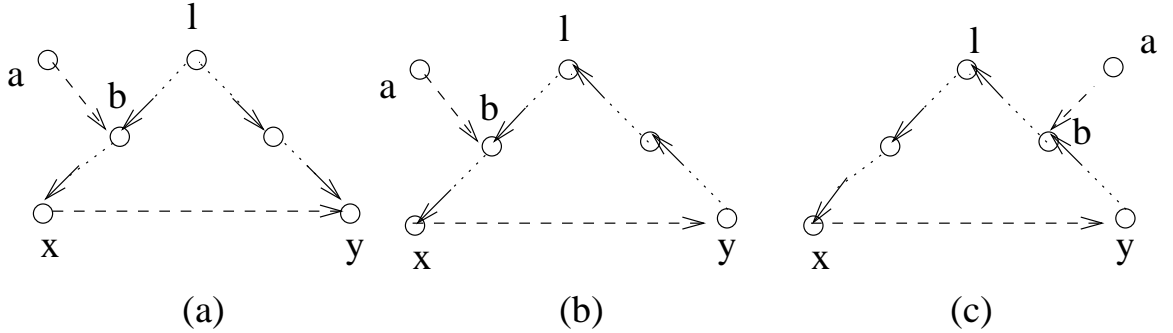


Figure 2: Illustrative cases for Claim I.

$LCA(c, d) - d$, there does not exist an edge (s, d) in G' . On the other hand, a path $s - c \rightarrow d$ exists in G_v . However, as may be observed from the diagram, if the shortest path is to pass through s and d , all the intermediate edges between s and d will be tree edges only, because tree edges have zero costs.

Next, consider claim II. There is an edge (f, d) with non-negative cost $c(c, d)$ in G' . But, there is no path of the form $f - c \rightarrow d$ in G_v . Implicitly, path $f - d$ can be traversed using tree edges only.

Proof of Claim I

Consider any two successive nontree edges (a, b) and (x, y) in $p(v_1, v_2)$. By definition of G' there is an edge (a, b) with cost $c(a, b)$ in G' . Let $LCA(x, y) = \ell$.

Now, if b lies between ℓ and x or ℓ and y as illustrated in Figure 2(a), 2(b) and 2(c), then there exists a valid directed path from a to y in G' using the two edges (a, b) and (b, y) with cost $c(a, b) + c(x, y)$ which is also the cost of the path $p(a, b)$ in G_v .

On the other hand, when b does not satisfy the above conditions, the following three cases can occur.

Case 1: Let b be an ancestor of ℓ and both the paths $\ell - x$ and $\ell - y$ are forward paths as illustrated by Figure 2(d). In this case, the path $a - b - x - y$ can not belong to any shortest path in G_v , as there is a zero cost path from

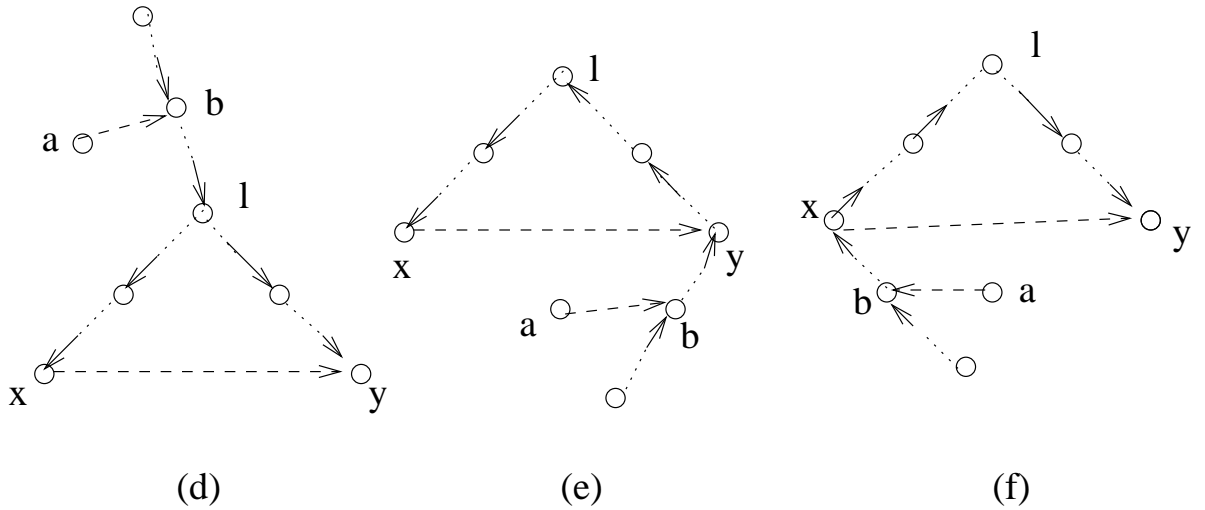


Figure 2: Illustrative cases for Claim I *continued*.

b to y using only tree edges .

Case 2: Let b lie below y and the path $y - \ell$ is a reversed path as illustrated by Figure 2(e). This is similar to Case 1 and clearly the path $a - b - x - y$ can't belong to any shortest path in G_v .

Case 3: Let b lie below x and the path $x - \ell$ is a reversed path as illustrated by Figure 2(f). This case is similar to the previous two cases.

Since we have considered any two successive nontree edges belonging to the path $p(v_1, v_2)$ in G_v , Claim I follows by induction.

Proof of Claim II

Consider the nontree edge (x, b) in G_v which caused the addition of the edge (a, b) in G' . By construction of G' , a must lie between ℓ and x or ℓ and y in G_v . Corresponding to the edge (a, b) in G' , there always exists a path from a to b in G_v either of same weight, or of zero weight (if it uses tree edges only). If there is a zero weight path in G_v from a to b , only the following three cases can occur.

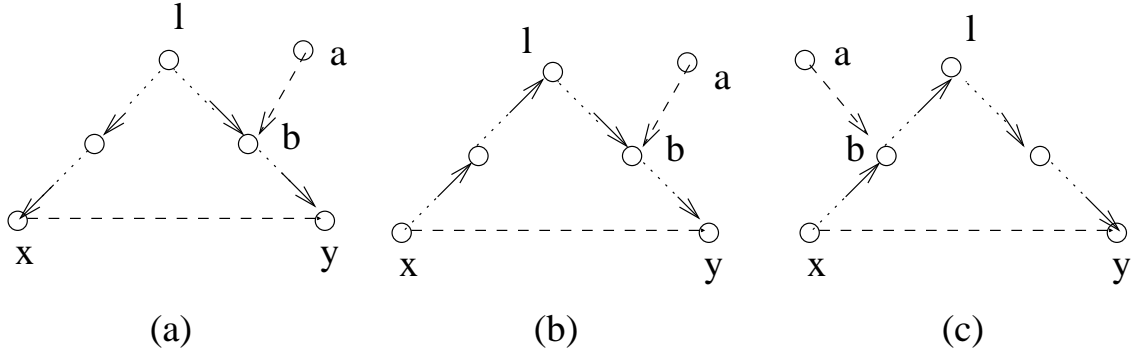


Figure 3: Illustrative cases for Claim II.

Case 1: Let a lie between ℓ and b and both the paths $\ell - x$ and $\ell - b$ are forward paths as illustrated by Figure 3(a). Let (a, b) belong to some shortest s -to- v path, say $p_S(s, v)$, in G' . As just noticed, $p_S(s, v)$ corresponds to a path in G_v . Further, both the paths $\ell - x$ and $\ell - b$ in G_v consist only of forward edges and by definition of G_v all edges from s to v have been reversed. This implies that v can't belong to the subtree rooted at ℓ in G_v . Therefore, $p_S(s, v)$ necessarily includes an edge (u, z) such that u belongs to the subtree rooted at b in G_v and z does not.

Let c_{in} be the cost of subpath $p(b, u)$ of the path $p_S(s, v)$ in G' . So the total cost of subpath $p(a, z)$ in G' is $c(a, b) + c_{in} + c(u, z)$. But notice that a also has to lie on the path $p(LCA(u, z), u)$ in G_v and therefore, there exists an edge (a, z) of cost $c(u, z)$ in G' . This implies that the cost of $p_S(s, v)$ can be reduced by traversing the edge (a, z) in G' . This is a contradiction, implying that $p_S(s, v)$ can't be a shortest path in G' .

Case 2: Let a lie between ℓ and b and the path $x - \ell$ is a reversed path as illustrated by Figure 3(b). Following similar arguments as given for the previous case it can be proved that (a, b) can't belong to a shortest s -to- v path in G' .

Case 3: Let a lie between ℓ and x and the path $x - \ell$ is a reversed path as illustrated by Figure 3(c). Similar arguments as in the previous cases prove the result.

From Claim II, shortest path from s to v in G' corresponds to a

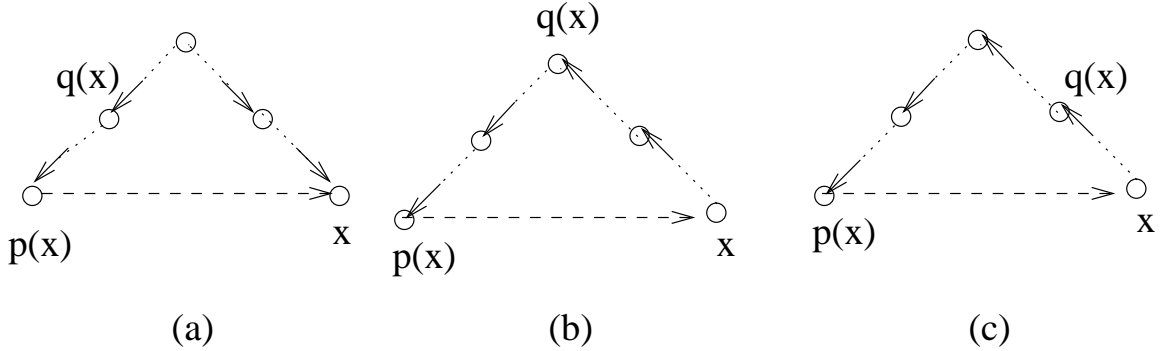


Figure 4: Reconstruction of paths.

valid path of same weight in G_v . But from Claim I the shortest path in G_v corresponds to a valid path (of same weight) in G' . If these two weights are different, a contradiction follows. Thus computing a shortest path in G' is equivalent to computing a shortest path in G_v .

4.1 Validity of path construction procedure

There are only three possibilities regarding relative positions of $p(x)$ and $q(x)$ for any vertex x in G_v , as illustrated by Figure 4.

Case 1: The first case occurs when the relative positions of $p(x)$ and $q(x)$ are as shown in Figure 4(a). In this case $q(x)$ does not belong to P_1 . Therefore, it can be safely included in any one of the paths P'_v or P''_v . Thus, one of the desired paths utilizes edges in P_2 only, while the other one uses the edges in P_1 .

Case 2: The next case occurs when $q(x) = \ell$ and $q(x)$ lies on P_1 as shown in Figure 4(b). Since the two paths P'_v and P''_v have to be edge-disjoint, one of them utilizes the edges in P_2 only, while the other uses the edges in P_1 .

Case 3: The last case occurs when $q(x)$ lies on P_1 and P_2 uses the reversed edges from $q(x)$ to ℓ as shown in Figure 4(c). The undirected path segment $q(x) - \ell$ is common to both the paths and has to be deleted. While traversing from $p(x)$ via its parents in T , $q(x)$ will never be encountered. Thus, deletion

of edges on the tree path $q(x)$ to ℓ will not matter for one of the desired paths. For the other path, the deleted segment of path can be avoided if $q(x)$ is reached through a nontree edge which has $q(x)$ as its head vertex.

5 Implementation of the Algorithm

Step 1 of the algorithm takes $n^3/\log n$ processors and runs in $O(\log^2 n)$ time [GB 86].

In Step 3 G' is constructed. A parallel implementation of this step can be done as follows.

Compute the path matrix F as in [TC 84]. This matrix provides the tree path from v to the root for each vertex v . The computation of F takes $O(\log n)$ time with $n^2/\log n$ processors. Once F is available the lowest common ancestor (LCA) for each pair of vertices defining a nontree edge can be computed in $O(\log n)$ time with one processor per edge. Several other efficient parallel algorithms for finding LCA s of a given set of vertex-pairs may also be found in the literature [SV 88].

Next, for each nontree edge (a, b) one processor is assigned for each vertex, say v , on the undirected tree path $a - LCA(a, b) - b$ and an edge (v, b) is added in G' with cost $c(a, b)$. On completion of this phase, G' may have multiple edges between v and b . Only the edge with minimum cost is retained and the remaining edges are deleted. Since there may only be a maximum of n multiple edges between any two vertices, this phase can be accomplished in $O(\log n)$ time using $n^3/\log n$ processors.

So, overall Step 3 needs $n^3/\log n$ processors and $O(\log n)$ time.

However, this step can also be implemented using $n^2/\log n$ processors in $O(\log n)$ time using a variant of tree contraction procedure [ADKP 89]. We just indicate the steps needed for constructing edges incident on a single vertex. The processor count needs to be multiplied by n to obtain the full graph.

Consider Figure 5. Vertex v has multiple incoming nontree edges.

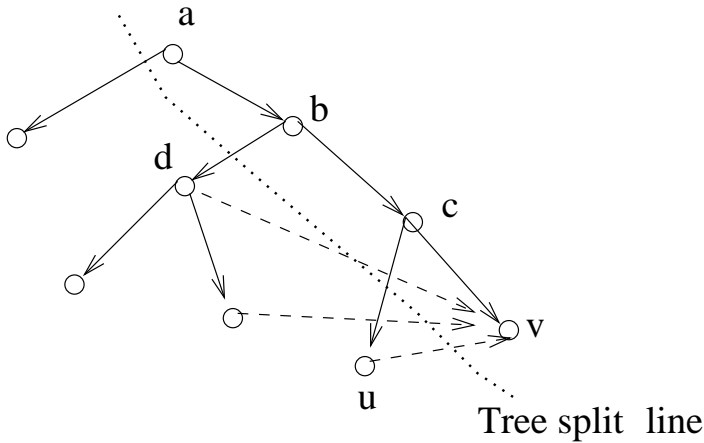


Figure 5: Applying tree contraction to construct G' .

Assign the cost $c(k, v)$ of each nontree edge with head v to its tail, i.e, vertex k . Assign infinite cost to all the remaining vertices.

Now if a tree contraction is carried out (at each vertex computing the minima of the costs of the children), apparently we obtain the minimum cost nontree edge for each vertex. However, this is incorrect as seen in the following example: if cost $c(u, v)$ is less than the cost $c(d, v)$ then the vertex b would choose the edge (u, v) while it actually does not lie in the path $u - LCA(u, v) - v$. We therefore make a minor modification to achieve the desired result.

Note that by using the tree contraction routine, correct values are obtained for vertices which lie along the tree path between the LCA and the two end-points of a given nontree edge. But for vertices lying above, a spurious value may be obtained. To get around this problem, the tree is split up into multiple subtrees as explained below.

Step 3.1: At each vertex on the path from v to the root of the tree, detach subtrees which don't contain v . Thus, we obtain a set of disjoint subtrees. The vertices which lie on the path from v to the root form a linked list, with the root as the head and v as the tail of the list. This partition of the tree is indicated by the dotted line labelled *Treesplitline* in Figure 5.

Step 3.2: Apply a tree contraction in each subtree. As is clear, each vertex

in the subtree gets its proper value.

Step 3.3: Each vertex in the linked list computes the minimum value in the roots of the subtrees that were detached from it.

Step 3.4 Compute prefix minima for the elements in the list.

This solves our problem, because any vertex k on the path from root to v necessarily lies on the *LCA* path of a nontree edge (with v as one endpoint) if this *LCA* is an ancestor of k .

Since there are a total of $O(n)$ vertices in all the subtrees, tree contraction runs in $O(\log n)$ time using $n/\log n$ processors. Also, prefix minima on a linked list has the same bounds [CV 89]. Overall, computation for edges terminating at a single vertex will need $n/\log n$ processors and $O(\log n)$ time. Hence, the construction of the entire graph, with n vertices, requires $n^2/\log n$ processors and $O(\log n)$ time.

The implementation of Step 4 is same as Step 1.

The reconstruction of the paths is done by Step 5. It will suffice to consider only the implementation of path reconstruction for one vertex v . The process can be replicated for multiple sinks as required.

Each vertex lying on the shortest path from s to v in T may be marked in $O(\log n)$ time with $n/\log n$ processors. After marking vertices, construct a temporary directed graph G'' whose vertex set is exactly the set of all vertices in G and the edges are defined as follows: each vertex x has a single outgoing edge - if x is marked it has an edge to $p(x)$ (defined earlier in Sec 3), else the edge is directed towards its parent in the shortest path tree T . G'' is actually a directed tree. Therefore, applying Euler tour on this tree all the vertices lying on the path from v to s can be found out. This provides one path. For the second path unmark all the marked vertices which lie on the first path and redefine their edges in G'' , i.e, for these vertices the new (and only) edge in G'' will be directed towards its parent in the shortest path tree T . Finally, perform another traversal from v to s to get this path. This step takes $O(\log n)$ time using $n/\log n$ processors for one vertex. Therefore, a total of $n^2/\log n$ processors is needed with the same time bound for all vertices.

The details of parallel implementation of each step of the multiple sink algorithm imply an overall time complexity of $O(\log^2 n)$ when $n^3/\log n$ processors are used.

The main bottleneck is shortest path computation which has a time complexity of $O(\log^2 n)$ with $n^3/\log n$ processors. All other steps run in $O(\log n)$ time using $n^2/\log n$ processors.

6 Conclusion

A closely related problem is finding vertex disjoint paths from a source to multiple sinks. This problem can be tackled following an approach quite similar to that used here for finding edge disjoint pair of shortest paths. An auxiliary graph G''' may be constructed from the given input graph G as follows:

Split each vertex v into v' and v'' and add an edge between them, with zero cost. In G if v has an incoming edge from x , in G''' create an edge from x'' to v' with cost $c(x,v)$. Now edge disjoint paths of G''' give vertex disjoint paths of G .

Appendix

The notations and terminology used here are same as in [AMO 93] .

In an unit capacity network, consider any feasible flow x of value v . The flow decomposition theorem (Theorem 3.5, [AMO 93]) states that any flow can be decomposed along paths and cycles. Since flows along cycles do not affect the flow value, the flows along the paths sum to v . Further, as capacity of each arc is 1, the paths are arc-disjoint and each path carries one unit of flow. This implies that the network has v arc-disjoint paths. Thus, a flow of value 2 from s to v with minimum cost solves the single-sink problem.

Next, we consider the following two theorems:

Theorem 1(*Theorem 9.3*, [AMO 93]) A feasible solution x^* is an optimal solution to the min-cost-flow problem iff some set of node potentials π satisfy the reduced cost optimality conditions:

$$c_{ij}^\pi \geq 0 \text{ for every arc } (i, j) \text{ in the residual network } G(x^*).$$

Theorem 2(*Lemma 9.12*, [AMO 93]) Suppose that a pseudoflow x satisfies the reduced cost optimality condition and we obtain x' from x by sending flow along a shortest path from node s to some other node k ; then x' also satisfies the reduced cost optimality conditions.

A zero flow, by definition, is a pseudoflow satisfying the reduced cost optimality conditions. So, if one unit of flow is sent along the shortest path from s to v , reduced cost optimality is maintained. After sending one unit of flow, since all arc-capacities are one, in the residual network the arcs along which flow is sent are reversed and all arcs which lie on the shortest path to some node have a reduced cost of zero. This is exactly G_v as defined in Sec 2 of this paper.

So, if another unit of flow can be sent in G_v along the shortest path from s to v , we obtain an optimal solution for the single-sink problem. Thus, only two iterations of shortest path algorithm are required to solve the single-sink problem.

Acknowledgements

The authors would like to thank the anonymous referees whose constructive suggestions improved the presentation of this paper. Most importantly, we owe the current format of the claims in the proof to one of them. The authors S. Banerjee and A. P. K. Reddy would like to take this opportunity to thank R. K. Ahuja who introduced them to the fundamental concepts of Network flows.

References

- [ADKP 89] K.Abrahamson, N.Dadoun, D.G.Kirkpatrick and T.Przytycka, A simple parallel tree contraction algorithm, *Journal. of Algorithms*, **10:2**(1989), pp 287-302.
- [AMO 93] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network flows : Theory, algorithms and applications*, Prentice-Hall, Englewood Cliff., NJ 1993.
- [AMOT 90] R. K. Ahuja, K.Mehlhorn, J. B. Orlin and R.E.Tarjan, Faster algorithms for the shortest path problem,*Journal of ACM*, **37**(1990), pp 213-223.
- [CV 89] R. Cole and U. Vishkin, Fast optimal parallel prefix sums and list ranking, *Information and Computation*, **81**(1989), pp 334-352.
- [Dij 59] E. W. Dijkstra, A note on two problems in connexion with graphs, *Numer. Math*, **1**(1959), pp 269-271.
- [FT 84] M.L.Friedman and R.E.Tarjan, Fibonacci heaps and their uses in improved network flow algorithms, *Proc. of 25'th annual IEEE. Symp. on FOCS*, (1984), pp 338-346.
- [GB 86] R. K. Ghosh and G. P. Bhattacharjee, Parallel algorithm for shortest paths, *IEE Proceedings-E: Computers and Digital Techniques*, **133**(1986), pp 87-93.
- [John 82] D.B.Johnson, A priority queue in which initialization and queue operations take $O(\log \log D)$ time, *Mathematical Systems Theory*, **15**(1982), pp 295-309.
- [ST 84] J. W. Suurballe and R. E. Tarjan, A quick method for finding shortest pairs of disjoint paths, *Networks*, **14**(1984), pp 325-336.
- [SV 88] B. Schieber and U. Vishkin, On finding lowest common ancestors: simplification and parallelization, *Proc. of 3rd Aegean Workshop on Computing, LNCS*, **319**(1988), pp 111-123.
- [TC 84] Y. H. Tsin and Y. F.Chin, Efficient parallel algorithms for a class of graph theoretic problems, *SIAM J. Comput.*, **13:3**(1984), pp 580-599.