M. Desrochers

A note on the partitioning shortest path algorithm

Department of Operations Research and System Theory Report OS-R8703 February

A Note on the Partitioning Shortest Path Algorithm

Martin Desrochers
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Recently Glover, Klingman and Philips proposed the Partitioning Shortest Path (PSP) algorithm. The PSP algorithm includes as variants most of the known algorithms for the shortest path problem. In a subsequent paper, together with Schneider, they proposed several variants of the PSP and conducted computational tests. Three of the variants were the first polynomially bounded shortest path algorithms to maintain sharp labels as defined by Shier and Witzgall. Two of these variants had computational complexity $O(|N|^2|A|)$, the other $O(|N|^3)$. In this note, we add a new step to the PSP algorithm resulting in new variants also scanning from sharp labels and having computational complexity $O(|N|^3)$ for two of them and $O(|N|^2)$ for the other. This new step also provides a test for the early detection of negative length cycles.

1980 Mathematics Subject Classification: 90C27,90C35,90C39,68R05,05C38
Key Words & Phrases: shortest path, computational complexity, negative length cycles detection.
Warning: This report will be submitted for publication to a journal.
Acknowledgement: This research was partly supported by a NATO Science Fellowship obtained through the Natural Sciences and Engineering Research Council of Canada.

Few problems have so many applications as the shortest path tree (SPT) problem. It is used in distance matrix calculation, vehicle routing, traffic equilibrium problems or as a step in the resolution of problems as assignment, matching, knapsack, generalized assignment. This explains the great interest the SPT has generated in the past years; see GALLO and PALLOTTINO [4] for a survey covering both the single shortest path tree problem and the all-pairs shortest path problem. This survey also contains a section on reoptimization procedures. In this note we will restrict ourself to the Partitioning Shortest Path (PSP) algorithm.

1. NOTATIONS AND BACKGROUND

Consider a directed network $G = (N,A)$ with node set N and arc set A. We denote the cardinality of N and A by $|N|$ and $|A|$. Let $l(i,j)$ denote the arc length for arc $(i,j)\in A$. We assume that the network contains no cycles of negative length. In Section 3, a test to verify this assumption will be presented. For a given node $r\in N$, which we call the root, we want to determine for each $v\in N$, the length of the shortest path from r to v, $d(v)$, and to build a directed tree rooted at r such that the unique path from the root to any other node is the shortest path between these nodes in G.

All algorithms for the shortest path tree have in common that they start from an arbitrary tree T and arbitrary labels $d(u)$. Usually the following initial labels are chosen:

$$d(r) = 0,$$

$$d(u) = \infty, \quad \forall u\in N,\ u\neq r.$$

The algorithm updates the tree T and the labels whenever it finds an arc $(i,j)\in A$ such that

$$d(i) + l(i,j) < d(j). \tag{1}$$

The label $d(j)$ is set to $d(i) + l(i,j)$, and the tree T is updated by replacing the current arc incident to j by the arc (i,j) . The process terminates when all arcs $(i,j)\in A$ satisfy Bellman's optimality conditions:

$$d(r) = 0, \tag{2}$$

$$d(i) + l(i,j) \geq d(j), \quad \forall (i,j) \in A. \tag{3}$$

GALLO and PALLOTTINO [3, 4] showed that all proposed algorithms are derived from this single prototype method. The only difference between algorithms is the data structure used to implement the search for arcs violating optimality condition (3). Usually all arcs going out of a node are treated consecutively, i.e. the algorithm scans a node. An extensive empirical study by DIAL *et al.* [2] has shown that no algorithm could be said to dominate all others on all problem instances. However, they concluded that, in general, for low density networks a label-correcting algorithm called $C2$ and proposed by PAPE [7] was the most efficient and that for networks of higher density a label-setting algorithm called $S2$ and proposed by DANTZIG [1] performs best. In an attempt to explain these results SHIER and WITZGALL [8] studied the properties of labelling algorithms. They discovered that Pape's algorithm has an exponential worst case complexity and that its successful behavior in practice could be explained by a property of the labels they called 'sharp'.

A node v has a sharp label if the length of the path from r to v in the current tree is equal to $d(v)$. In Figure 1, there is a five-node network. With each node is associated a label in parentheses and with each arc a length in brackets. The arcs belonging to the current tree are in bold, other arcs are dashed. Presently nodes 1,2 and 3 have sharp labels and nodes 4 and 5 do not have sharp labels because arc (2,3) was added to the current tree and $d(3)$ was updated. This tree modification causes labels $d(4)$ and $d(5)$ to be non-sharp. Any scanning of a node v with non-sharp label will have to be done again when $d(v)$ will be updated. It should be noted that for non-negative arc length networks label-setting algorithms always scan sharp nodes.

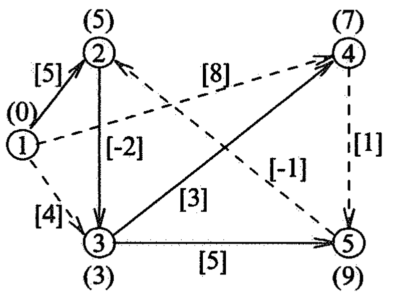

Figure 1. Examples of sharp and non-sharp labels.

GLOVER *et al.* [5] proposed the general *partitioning shortest path* (PSP) scheme. This scheme can be used to produce several polynomially bounded shortest path algorithms. In fact most algorithms proposed to solve the SPT can be interpreted as variants of the PSP.

THE PSP ALGORITHM (GLOVER *et al.* [5])

Step 0. Initialization

Initialize the predecessor $p(i)$ to define an arbitrary tree, and initialize a distance label $d(i)$ for each node:

$$p(i) = 0, \quad \forall i \in N,$$
$$d(i) = \infty, \quad \forall i \in N, \; i \neq r,$$
$$d(r) = 0.$$

Set iteration count $k = 0$. The set of scan eligible nodes will be partitioned in two lists NOW and NEXT. Initially NOW $= \{r\}$ and NEXT $= \emptyset$.

Step 1. Select an element of NOW.
If NOW is empty, go to step 3. Select any node u from NOW.

Step 2. Scan selected node u.
Delete node u from NOW. For each successor of u, i.e., $v \in \{v \mid (u,v) \in A\}$, if $d(u) + l(u,v) < d(v)$ then set $d(v) = d(u) + l(u,v)$, update T by setting $p(v) = u$ and add v to the NEXT list if v is not already in NEXT or NOW. When all successors have been examined go to step 1.

Step 3. Repartition scan eligible nodes.
If NEXT is empty, stop. (Bellman's conditions are met by all arcs.) Otherwise, set $k = k + 1$, transfer all nodes from NEXT to NOW and return to step 1.

In a subsequent paper, Glover *et al.* [6] proposed several variants of the PSP and in an empirical study compared them to the $C2$ and $S2$ algorithms. Three of the variants were the first polynomially bounded sharp label-correcting algorithms for the SPT. However, their computational complexity is an order of magnitude greater than their non-sharp equivalent. These variants have computational complexity $O(|N|^2|A|)$ for two of them while similar non-sharp algorithms have computational complexity $O(|N||A|)$. The third sharp algorithm has complexity $O(|N|^3)$. To overcome this disadvantage, they defined two near-sharp algorithms for the non-negative arc length case. In a near-sharp algorithm, at the beginning of each iteration all the nodes in NOW have sharp labels. Those near-sharp algorithms have computational complexity $O(|N||A|)$ and one of the two, THRESH-X2, outperforms other algorithms in the empirical study.

2. A new sharp algorithm for SPT

In order to scan only sharp nodes, the sharp algorithm by Glover *et al.* [6] maintains sharp labels at all the nodes through the whole process. This can be viewed as a specialized variant of the primal simplex algorithm. Each time the label of a node v is updated, the algorithm also updates the labels of all the nodes in the current subtree rooted at v. This represents a considerable computational burden.

However, to obtain a sharp algorithm, it is not necessary to maintain sharp labels at all the nodes; we only need to assure that the label of the node scanned is sharp. Checking if the label of a node is sharp is simpler than maintaining sharp labels at all the nodes. If the label of the node to scan is not sharp, we simply have to update this label first to have a sharp label. The following step 1B checks if a label is sharp and updates it (if necessary). This step can be inserted between step 1 and 2 in the PSP algorithm.

Step 1B. Check if the label of the current node u is sharp.

```
$v = u$,
$\delta = d(u)$,
while $v \neq 0$ do
    $\delta = \delta - l(p(v),v)$,
    $v = p(v)$,
end while;
{If $\delta = 0$, the label $d(u)$ is sharp;
  otherwise $\delta$ is the correction needed to obtain a sharp label. }
$d(u) = d(u) - \delta$.
```

Each Step 1B has computational complexity $O(|N|)$ if the access to $l(p(v),v)$ takes constant time. As the resulting algorithm is a variant of the PSP algorithm, Lemma 1,2 and Theorem 1 (except the part on computational complexity) of Glover *et al.* [5] hold. Adding step 1B modifies the computational complexity of the algorithm . By Lemma 1 and 2 there are still at most $|N|-1$ iterations of step 3. The maximum number of arcs treated at each iteration is still $|A|$. At each of the possible $|N|-1$ iterations, we have to check at most $|N|-1$ labels to see if they are sharp. Therefore this variant of the PSP has computational complexity $O(|N|^3)$.

The implementation of this variant is rather simple. We need three $|N|$ length arrays: the distance function, $d(u)$, the predecessor function, $p(u)$, and the array *lgtprec*(u) containing the length of the arc $(p(u),u)$. This last array enables constant time access to $l(p(u),u)$. The implementation of the sharp algorithm of GLOVER *et al.* [6] utilizes at least five $|N|$ length arrays: distance and predecessor functions as in our implementation and thread, depth and reverse thread functions to update the labels of the nodes in the subtree efficiently. Our simpler implementation and better time bound raise the hope that our variant of the PSP algorithm runs quicker than the other sharp algorithms.

3. A TEST TO DETECT NEGATIVE LENGTH CYCLES

The assumption that there is no negative length cycles is necessary to guarantee the existence of a shortest path tree. However, except for networks with non-negative arc length, the only way to detect negative length cycles is to use an SPT algorithm, say the PSP algorithm. If the algorithm terminates in less than $|N|$ iterations, then there is no negative length cycle, otherwise there is at least one. It is possible to use a variant of step 1B to detect negative length cycles earlier. If at any step 1B, the path from node v to root r in the current tree has more than $|N|$ arcs, there must be a negative length cycle. As the complete tree has exactly $|N|-1$ arcs, this is impossible and we have detected a negative length cycle.

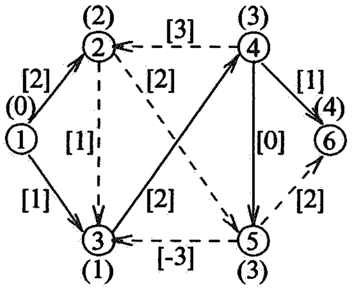

Figure 2.
Current tree before
scan of node 5.

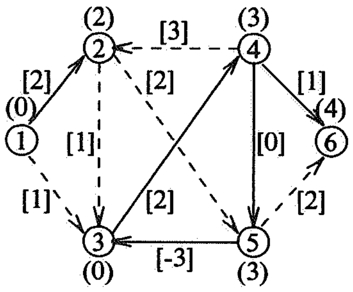

Figure 3.
Current tree after
scan of node 5.

In Figure 2, using the same conventions as in Figure 1, we can see the current tree before scanning of node 5 and formation of a cycle in this *tree*. As soon as the algorithm tries to scan a node belonging to the same *subtree* as the cycle, i.e. nodes 3,4,5,6 in Figure 3, the modified step 1B detects the cycle. The modified step 1B is as follows:

Step 1B. Check if the label of the current node u is sharp and try to detect negative length cycles.

```
$v = u$,
$lp = 0$,    { *is the* # of arcs in the path from $u$ to $r$ in the tree; }
$\delta = d(u)$.
While $v \neq 0$ and $lp < |N|$ do
    $\delta = \delta - l(p(v), v)$,
    $v = p(v)$,
    $lp = lp + 1$,
end while.
If $lp = |N|$ then *stop*. { Step 1B has detected a cycle. }
$d(u) = d(u) - \delta$.
```

Step 1B detects negative length cycles without modification to the computational complexity of the new algorithm.

4. Conclusion

In this note, a sharp PSP algorithm for the general case of the SPT was presented. It is possible to modify the two other sharp PSP algorithms presented by Glover *et al.* [6] in the same fashion to obtain a sharp treshold algorithm for the non-negative arc length case having computational complexity $O(|N|^3)$ or by limiting the number of nodes transferred from NEXT to NOW as in their third variant to obtain an algorithm having computational complexity $O(|N|^2)$.

We proved that sharp algorithms having computational complexity of the same magnitude as non-sharp algorithms exist, and devised a test for early detection of negative length cycles. Two questions remain open. Is there a sharp label-correcting algorithm with computational complexity $O(|N||A|)$? And will the improved sharp algorithms be competitive in practice?

References

[1]G.B. Dantzig (1963). *Linear Programming and Extensions.* Princeton University Press, Princeton,NJ.

[2]R. Dial, F. Glover, D. Karney and D. Klingman (1979). A computational analysis of alternative algorithms and labelling techniques for finding shortest path trees. *Networks 9,* 215-248.

[3]G. Gallo and S. Pallottino (1984). Shortest path methods in transportation models. M. Florian (ed.). *Transportation planning models,* North-Holland, Amsterdam, 227-256.

[4]G. Gallo and S. Pallottino (1986). Shortest path methods: a unifying approach. *Math. Programming Study 26,* 38-64.

[5]F. Glover, D. Klingman and N. Phillips (1985a). A new polynomially bounded shortest path algorithm. *Operations Res. 33,* 65-73.

[6]F. Glover, D. Klingman, N. Phillips and R. Schneider (1985b). New polynomial shortest path algorithms and their computational attributes. *Management Sci. 31,* 1106-1128.

[7]U. Pape (1974). Implementations and efficiency of Moore-algorithms for the shortest route problems. *Math. Programming 7,* 212-222.

[8]D. Shier and C. Witzgall (1981). Properties of labelling methods for determining shortest path trees. *J. Res. Nat. Bur. Standards 86,* 317-330.