A Multi-source Shortest Path Algorithm

Raffaele Cerulli* Stefano Pallottino†

*Department of Mathematics and Computer Science
University of Salerno
P.te Don Melillo, 84084, Fiscianop (SA) Italy
raffaele@unisa.it

†Department of Computer Science
Unicersity of Pisa
Via F. Buonarroti, 2 – 56127, Pisa, Italy
pallo@di.unipi.it

1 Introduction

In many transportation problems, it is necessary to compute many times shortest paths among particular nodes (denoted in general as *centroids*) in a directed and weighted graph $G = (N, A, c)$, where $n = |N|$ is the number of nodes, $m = |A|$ is the number of arcs, and c_{ij} denotes the *cost* of arc $(i,j) \in A$. By R we denote the set of centroids, of cardinality $k = |R|$.

The k to k shortest path problem, which we call for short as "k^2sp" problem, can be solved by working out k "shortest path tree" problems, one for each centroid either as origin or as destination of the paths [1]. An alternative approach to solve k^2sp problem, especially when $k = O(n)$, is to solve the "all pairs shortest path" problem, which requires to handle an n-order matrix of distances [5]. This memory requirement, in general, prevents to use such a kind of approach since the current size of transportation networks reaches thousands of nodes and arcs (for some problems the size of the network can reach million of nodes).

Another approach proposed in literature is based on the reoptimization approach, where one can exploit the optimal tree found for the i-th centroid to compute the shortest path tree of the i+1-th centroid [4, 6, 8].

In this paper we aim to show how it is convenient to simultaneously compute *k* shortest path trees (or a portion of them in case of lack of memory), by proposing a general approach, which we call *multisource*, that combines simplicity of the algorithm with its efficiency. The first experimental results obtained show how promising the *multisource* approach is, and suggest to deeply investigate the *multisource* shortest path algorithms efficiency to provide a powerful tool for transportation models and software. In fact, in principle, it is possible to generalize any existent shortest path tree algorithm to a similar *multisource* algorithm.

2 The Problem

Let $r \in R$ be a centroid, which will be the *root* of the finding shortest path tree, formed by paths having *r* as origin (the case in which *r* is the path destination can be symmetrically stated); the shortest path tree problem with root *r* can be written as a Linear Programming flow problem:

$$(P_r) \qquad \text{Min} \quad \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\sum_{(j,i)\in BS(i)} x_{ji} - \sum_{(i,j)\in FS(i)} x_{ij} = b_i \qquad \forall i \in N$$

$$x_{ij} \geq 0 \qquad \forall (i,j) \in A$$

where $FS(i) = \{(u,v) \in A : u = i\}$ and $BS(i) = \{(u,v) \in A : v = i\}$ denote the *forward star* and the *backward star* of $i \in N$, respectively; where the right hand side has values $b_r = -n+1$, and $b_i = 1, \forall i \neq r$, and the arc flow x_{ij}, when positive, gives the number of nodes in the subtree having *j* as root.

The dual problem of (P_r) is

$$(D_r) \qquad \text{Max} \quad (1-n)\pi_r + \sum_{j\neq r} \pi_j$$

$$\pi_j - \pi_i \leq c_{ij} \qquad \forall (i,j) \in A$$

where π_i denotes the dual variable associated with $i \in N$, usually referred to as the *potential* of i.

The dual constraints correspond to the so called Bellman conditions when a label array d is used as an estimation of the potential array:

$$d_i + c_{ij} \leq d_j \qquad \forall (i,j) \in A \tag{1}$$

To solve the k^2-sp problem we have to solve the collection of k pairs (P_r) and (D_r) problems, one for each root $r \in R$.

As far as the solution of a single pair (P_r) and (D_r) for a given $r \in R$ is concerned, the iterative part of any generic (primal) shortest path tree algorithm is depicted in figure 1. Q is the set of candidate nodes formed by all the nodes whose forward star may contain arcs violating the Bellman conditions (1). Moreover, p_j denotes the predecessor node of j, that is the node which precedes j in the current tree, for each $j \in N$. For more details see [1, 7].

```
repeat
   select and extract i from Q;
   foreach (i,j) ∈ FS(i) do
      if d_i + c_ij < d_j then
         d_j ← d_i + c_ij
         p_j ← i
         if j ∉ Q then insert j into Q
until Q = ∅;
```

Figure 1. The iterative part of a generic algorithm for a single problem

3 The multi-source approach

Let us suppose that we have enough memory to handle all the k shortest path trees; that is we have enough memory of order O(kn). In case of insufficient available memory, a subset of trees is simultaneously treated and the sequencing of trees to treat will be explained afterwards.

By $d^r{}_i$ and $p^r{}_i$ we denote the label and the predecessor of node $i \in N$, respectively, while by Q^r we

denote the set of candidate nodes, all relative to the root $r \in R$. Among the k roots, one of them, which we will denote from now as r, is chosen as the *leader root*, while the others roots $s \in R \setminus \{r\}$ are denoted as the *active* ones.

The basic idea is to apply a classical primal algorithm for the tree rooted at r and, each time a node i is extracted from Q^r, the algorithm simultaneously checks the Bellman conditions for all arcs $(i,j) \in FS(i)$ to all active roots s such that $i \in Q^s$, as shown in figure 2.

The leader root r obviously belongs to $R(i)$, so node i is removed from all the sets of candidate nodes. At the end of the iterative part, that is when $Q^r = \varnothing$, an active root is designed as new leader r, and the iterative part is repeated for that new leader. The *multisource* algorithm ends when all the roots have been treated as leader.

The advantage of such an approach resides in starting to work on a new leader root with a set of labels whose values are closer to the optimal values than the initial $+\infty$ values. The more important advantage is due to a better handling of the input data. In fact, once obtained the current node i, the selection from the memory of the data relative to $FS(i)$ requires a significant time of accessing to the graph data structures, which can be higher than the repeated checking of the Bellman conditions of arcs $(i,j) \in FS(i)$ for the active roots belonging to $R(i)$.

```
repeat
select i from Q^r;
set R(i) = {s ∈ R: i ∈ Q^s};
foreach s ∈ R(i) do extract i from Q^s;
foreach (i,j) ∈ FS(i) do
   foreach s ∈ R(i) do
      if d^s_i + c_ij <  d^s_j then
         d^s_j ← d^s_i + c_ij
         p^s_j ← i
         if j ∉ Q^s then insert j into Q^s
until Q^r = ∅;
```

Figure 2. The iterative part of a generic *multisource* algorithm

To implement a *multisource* algorithm some choices must be made:

- to define the order of roots to become leader;
- which type of classical algorithm is adopted for the leader root;
- how to implement the collection of sets of candidate nodes;
- in case of lack of memory, how to separate the active roots from the “not already activated”

ones, and how to activate them.

In the paper we will propose various solutions to the above points and we will investigate the more effective choices.

4 A particular *multisource* implementation and its performance

In this section we want to analyze the first experimental results for a very simple implementation of *multisource*. In particular:

- we use the increasing order of integer numbers as the roots order;
- we use, for the leader root, the algorithm Dijkstra with the 4heap [2];
- we implement a $k \times n$ bit-array B to describe the sets of candidate nodes for the active roots: $B[s,i] = 1$ if $i \in Q^s$, and 0 otherwise, for each $s \in R \setminus \{r\}$ and for each $i \in N$;
- we restrict the test graph size to treat all the roots simultaneously.

In the following we will call *MS-D4h* the implemented algorithm and we will compare its performance versus its twin algorithm *D4h*, that is the classical Dijkstra algorithm with a 4heap as a priority queue [2, 3].

In table 1 we report the results obtained for a complete graph with n=3000 nodes (and almost 9 million of arcs), and with arc costs uniformly generated in the interval [1, 10000]. We vary the number of roots (randomly chosen) from 10% to 100% of the nodes.

For the tests we used a PC AMD Athlon Thunderbird 1.2 GHz, 512 MB Dimm Sdram PC133 with Suse Linux 8.2. The algorithms have been coded in C language and compiled with gcc ver.3.3 with option “-o4”.

The c.p.u. times reported are relative to the computational part only, that is the time to read the graph and to write the results is not computed.

Column 2 reports the global number of node extractions from the set of candidate nodes Q for the algorithm *D4h*. In column 3 there is the global number of node extractions from the various Q^r for the algorithm *MS-D4h* when r is the leader root, while in columns 4 it is reported the global number of forward stars analyzed. Columns 5 and 6 give the total running time in seconds. Finally, the last column gives the rate of the running time saving obtained with the *multisource* approach.

For both algorithms we stop the single root calculation only when the set of candidate nodes is empty; while, it is well known that, when the Dijkstra approach is used for a graph with non-negative arc costs, it is possible to stop the computation once the last centroid is extracted from Q.

Table 1: Experimental results

# Roots ***% n***	**Heap Extr** *D4h*	**Heap Extr** *MS-D4h*	**# FS analyz.** *MS-D4h*	**TIME** *D4h*	**TIME** *MS-D4h*	**% SPEED UP**
10	900,000	12,848	1,606,665	128.78	119.13	8.10 %
20	1,800,000	13,317	3,226,702	257.75	229.35	12.36 %
30	2,700,000	13,527	4,841,978	386.58	351.73	9.89 %
40	3,600,000	13,637	6,442,847	515.52	445.65	15.72 %
50	4,500,000	13,724	8,063,571	644.32	555.42	16.00 %
60	5,400,000	13,802	9,680,413	773.20	658.28	17.45 %
70	6,300,000	13,842	11,304,479	902.08	778.78	15.83 %
80	7,200,000	13,926	12,913,818	1,030.97	882.10	16.87 %
90	8,100,000	14,056	14,528,160	1,159.88	1,000.68	15.90 %
100	9,000,000	14,077	16,156,401	1,288.80	1,109.45	16.16 %

The preliminary results are very promising. They show that the *multisource* approach is effective. This is mainly due to the number of accesses to the graph structure to select the nodes forward stars. In fact, in the case of 100% of nodes playing the role of roots, the number of accesses decreases from 9 millions to few thousands, see columns 2 and 3. The saving due to that drastically reduced number of accesses to the graph structure compensates the overhead due to the higher number of scans of the forward stars, see column 4, since only the leader root computation follows the Dijkstra's ordering of nodes selection.

Indeed, if the node label d^s_j, of a node j related to a root s, reaches the optimum value before that s becomes the leader root, then there is no need of using heap insertion, successive extraction and forward star analysis for node j for that root.

In case of lack of memory, two approaches are possible:

- *refreshing*: every time that the computation for the leader root is completed, a new root is activated and stored in place of the leader root, and another root is selected as leader;
- *paging*: all the active roots are processed, without substitutions. Once completed the computation for all of them, an equal number of new roots is activated, and one of them is selected as leader.

5 Conclusions

The preliminary experimental results suggest to perform a wide experimentation of the multisource approach, both by increasing the number of graph tests (with different structure and dimension) and by producing different versions of multisource.

In fact, to each classical shortest path algorithm, a multisource counterpart can be paired; this allows to deeply investigate which type of data structures will result more effective to exploit the multisource approach.

The presentation at the Conference will be devoted to the analysis of the experimentation.

After a couple of days this extended abstract was concluded, Prof. Stefano Pallottino suddenly died.
I want to dedicate this paper to his memory to express my gratitude for all the time spent working together.

References

[1] R. K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall Englewood Cliffs, NJ, 1993.

[2] F. Carrabs and R. Cerulli, The Shortest Path Problem: Algorithm and Data Structures, University of Salerno, Department of Mathematics and Computer Science, *Tech. Rept.* N. 16, 2003.

[3] B. V. Cherkassky, A. V. Goldberg and T. Radzik, Shortest paths algorithms: Theory and experimental evaluation', *Mathematical Programming* 73, 129-174 (1996).

[4] M. Florian, S. Nguyen and S. Pallottino, A dual simplex algorithm for finding all shortest path, *Networks* 11**,** 367-378 (1981).

[5] R. W. Floyd, Algorithm 97: Shortest path, *Communications of the A.C.M.* 5, 345 (1962).

[6] G. Gallo and S. Pallottino, A new algorithm to find the shortest paths between all pairs of nodes, *Discrete Applied Mathematics* 4, 23-35 (1982).

[7] G. Gallo, and S. Pallottino, Shortest path algorithms, in (B. Simeone et al., eds.) *Fortran codes for network optimization***,** *Annals of Opeations. Research* 13, 3-79 (1988).

[8] S. Nguyen, S. Pallottino and M. G. Scutellà, A new dual algorithm for shortest path reoptimization, in *Transportation and Network Analysis: Current Trends*, M. Gendreau and P. Marcotte (eds), Kluwer (2002) 221-235.