

Distributed Negative Cycle Detection Algorithms *

Luboš Brim, Ivana Černá and Lukáš Hejtmánek

Faculty of Informatics, Masaryk University

Brno, Czech Republic

{brim, cerna, xhejtman}@fi.muni.cz

Summary: Several new distributed algorithms for the single source shortest paths and for the negative cycle detection problems on arbitrary directed graphs given by adjacency list are developed, theoretically analysed, proved correct, and experimentally compared. The algorithms are to be performed on clusters of workstations that communicate via a message passing mechanism.

1 Introduction

The single source shortest path problem (SSSP) is a fundamental problem with many theoretical and practical applications and with several effective and well-grounded sequential algorithms. The same can be said about the closely related negative cycle problem (NCP) which is to find a negative length cycle in a graph or to prove that there are none. In fact, all known algorithms for NCP combine a shortest paths algorithm with some cycle detection strategy.

In many applications we have to deal with extremely large graphs (a particular application we have in mind is briefly discussed below). Whenever a graph is too large to fit into memory that is randomly accessed a memory that is sequentially accessed has to be employed. This causes a bottleneck in the performance of a sequential algorithm owing to the significant amount of paging involved during its execution. A usual approach to deal with these practical limitations is to increase the computational power (especially randomly accessed memory) by building a powerful (yet cheap) distributed-memory cluster of computers. The computers are programmed in single-program, multiple-data style, meaning that one program runs concurrently on each processor and its execution is specialised for each processor by using its processor identity (id). Parallel programs rely on communication layer based on Message-Passing Interface standard.

Our motivation for this work was to develop a distributed model checking algorithm for linear temporal logic[BČKP01b] which can be reduced to the negative cycle problem.

*This work has been partially supported by the Grant Agency of Czech Republic grant No.201/03/0509.

In this particular application the graph to be processed is not completely given at the beginning of the computation through its adjacency-list or adjacency-matrix representation. Instead, we are given a source vertex together with a function which for every vertex computes its adjacency-list. A possible approach is to generate the graph at first and then to process it with a distributed SSSP resp. NCP algorithm. However, this approach is highly non-efficient. If one processes the graph simultaneously with its formation it can happen that a negative cycle is detected even before the whole graph is formatted. Moreover, this *on-the-fly* technique allows to generate only the part of the graph reachable from the source vertex and thus reduces the space requirements for the graph representation. As successors of a vertex are determined dynamically there is no need to store any information about edges permanently which brings yet another reduction in space complexity.

A natural starting point for building a distributed algorithm is to distribute an efficient sequential algorithm. Because of the aforementioned reasons we have concentrated on algorithms which admit graphs specified with the help of adjacency lists (and omit those presupposing adjacency matrix representation of the graph). These algorithms (for an excellent survey see [CG99]), which are based on relaxation of graph's edges, are inherently sequential and their parallel versions are known only for special settings of the problem. For general digraphs with non-negative edge lengths parallel algorithms are presented in [MS00, RV92, CMMS98] (see [HTB98] for a comparative study) together with studies concerning good decomposition [HTB97]. For special cases of graphs, like planar digraphs [TZ96, DKZ94], graphs with separator decomposition [Coh96] or graphs with small tree-width [CZ95] more efficient algorithms are known. Yet none of these algorithms are applicable on general digraphs with potential negative-length cycles.

In this paper we propose several distributed algorithm for the general SSSP and NCP problems on graphs with integer edge lengths and given by adjacency lists, we analyse their worst-case complexity and we conduct an extensive practical performance study of these algorithms. We study various combinations of distributed shortest paths algorithms and distributed cycle detection strategies to determine the best combination measured in terms of *scalability*.

2 Serial Negative Cycle Problem

We are given a triple (G, s, l) , where $G = (V, E)$ is a directed graph with n vertices and m edges, $l : E \rightarrow R$ is a *length function* mapping edges to real-valued lengths, and $s \in V$ is the *source* vertex. The *length of the path* $\rho = \langle v_0, v_1, \dots, v_k \rangle$ is the sum of the lengths of its constituent edges, $l(\rho) = \sum_{i=1}^k l(v_{i-1}, v_i)$. *Negative cycle* is a cycle $\rho = \langle v_0, v_1, \dots, v_k, v_0 \rangle$ with length $l(\rho) < 0$. The *negative cycle problem* is to find a negative cycle in a graph or to prove that there are none.

Algorithms for the negative cycle problem using the adjacency-list representation of the graph construct a shortest-path tree $G_s = (V_s, E_s)$, where V_s is the set of all vertices reachable from the source, $E_s \subseteq E$, s is the root of G_s and for every vertex $v \in V_s$ the path from s to v in G_s is the shortest path from s to v in G .

The labeling method maintains for every vertex v its *distance label* $d(v)$ and *parent* $p(v)$. Initially $d(v) = \infty$ and $p(v) = \text{null}$, the method starts by setting $d(s) = 0$. The method maintains for every vertex its status which is either *unreached*, *labeled*, or *scanned*, initially all vertices but the source are *unreached* and the source is *labeled*. The method and is based on the scan operation. During scanning a vertex v the edges out-coming from v are *relaxed* which means that if $d(u) > d(v) + l(v, u)$ then $d(u)$ is set to $d(v) + l(v, u)$ and $p(u)$ is set to v . The status of v is changed to *scanned* while the status of u is changed to *labelled*. During the computation the edges $(p(v), v)$ for all $v : p(v) \neq \text{null}$ induce the *parent graph* G_p . If all vertices are either *scanned* or *unreached* then d gives the shortest path lengths and G_p is the shortest-path tree. On the contrary, any cycle in G_p is negative and if the graph contains a negative cycle then after a finite number of scan operations G_p always has a cycle. This fact is used for the detection of negative cycle detection.

2.1 Scanning strategies

Different strategies for selecting a *labeled* vertex to be scanned next lead to different algorithms.

The *Bellman–Ford–Moore algorithm* [Bel58, ?] uses for selecting the FIFO strategy and runs in $\mathcal{O}(nm)$ time.

The *D’Escopo–Pape algorithm* [Pap74] makes use of a queue. The next vertex to be scanned is removed from the head of the queue. A vertex that becomes labeled is added to the head of the queue if it had been scanned previously, or to the tail otherwise.

The *Pallotino’s algorithm* [Pal84] maintains two queues. The next vertex to be scanned is removed from the head of the queue if it is nonempty and from the second queue otherwise. A vertex that becomes labeled is added to the tail of the first queue if it had been scanned previously, or to the tail of the second queue otherwise. Both last mentioned algorithms favour recently scanned vertices and runs in $\mathcal{O}(n^2m)$ time in the worst case, assuming no negative cycles.

The *network simplex algorithm* [Dan51] maintains the invariant that in the current parent graph all edges have zero reduced cost (the reduced cost of the edge (v, u) is $l(v, u) + d(u) - d(v)$). Therefore if the distance label of a vertex u decreases, the algorithm decreases labels of vertices in the subtree rooted at v by the same amount. Then a new edge with negative reduced cost (so called *pivot*) is found and the process continues. There are several heuristics to find a pivot. We can search the scanned vertices and choose the pivot according to a FIFO strategy or depending on the value of the reduced cost. The algorithm runs in $\mathcal{O}(nm)$ time. DOPLNIT PODROBNEJSIE????

The Goldberg–Radzik algorithm [GR93] vyuziva topologicke triedenie a jeho distribuo- vanuv verziu sme neuvazovali. DOPISAT????

2.2 Cycle Detection strategies

Besides the trivial and non-efficient cycle detection strategies like time out and distance lower bound, the algorithms can use one of the following strategies: *walk to the root*,

subtree traversal and subtree disassembly.

The *walk to root* method tests whether G_p is acyclic. Suppose the scanning operation applies to an edge (v, u) (i.e. $d(u) \geq d(v) + l(v, u)$) and the parent graph G_p is acyclic. This operation will create a cycle in G_p if and only if u is an ancestor of v in the current tree. Before applying the operation, we follow the parent pointers from v until we reach u or s . If we stop at u we have found a negative cycle; otherwise, the scanning operation does not create a cycle. The walk to root method gives immediate cycle detection. However, since the path to the root can be long, the cost of applying the scanning operation to an edge becomes $\mathcal{O}(n)$ instead of $\mathcal{O}(1)$. In order to optimise the overall computational complexity we propose to use amortisation to pay the cost of checking G_p for cycles. More precisely, the parent graph G_p is tested only after the underlying algorithm performs $\Omega(n)$ work. The running time is thus increased only by a constant factor. The correctness of the amortised strategy is based on the fact that if G contains a negative cycle reachable from s , then after a finite number of scanning operation G_p always has a cycle [CG99]. ASI PRIPSAT, ZE VRCHOLY JE POTREBA ZNACIT/ZAMYKAT

The *subtree traversal* method makes use of a symmetric idea: scanning of an edge (v, u) can create a cycle in G_p if and only if v is an ancestor of u in the current tree. This strategy fits naturally with the network simplex method as the subtree traversal can be combined with the updating of the pivot subtree.

The *subtree disassembly* method also searches the subtree rooted at u . However, this time if v is not in the subtree, all vertices of the subtree except u are removed from the parent graph and their status is changed to *unreached*. The work of subtree disassembly is amortized over the work to build the subtree and cycle detection is immediate.

3 Distributed Negative Cycle Detection Algorithms

3.1 Distributed scanning strategies

We are providing distributed versions of all the algorithms except Goldberg–Radzig. The scanning strategies can be used either in the distributed environment at no cost. All provided algorithms have the same asymptotic complexity as sequential ones.

The distributed version of the Bellman–Ford–Moore algorithm is straight forward. Correctness of this algorithm does not depend on the sequence of relaxations. We only cannot relax one vertex two times at once. This property is kept because the vertices are distributed into disjoint sets and all algorithms run sequentially at one processor.

The heuristics D’Escopo–Pape and Pallotino do not need any modification at all. They only change the sequence of relaxations that does not change the correctness.

The network simplex algorithm uses the same selection rule for relaxing as the Bellman–Ford–Moore algorithm. We need to provide only distributed subtree traversal to maintain the invariant about zero reduced cost. However subtree traversal is just breadth first traversal of the parent graph that is easy to distribute. The parent graph does not contain the cycle unless original graph contains reachable negative cycle.

3.2 Distributed Cycle Detection strategies

All three considered cycle detection strategies are usable in the distributed environment without the change of their complexity.

The subtree traversal method needs just a little change. The breadth first search in the parent tree can be distributed in natural way. We only have to check if the found cycle is really the negative cycle, not every found cycle have to be the negative one because of asynchronous relaxations. MOZNA VICE VYSVETLIT????

The more complicated is the subtree disassembly. The correctness of sequential version cannot be hold while we do no vertex locking. We do not have to find some very special negative cycle when all instances of algorithm do strictly synchronous sequence of relaxations and subtree disassemblies. Another strategy is need to hold the correctness. We have used the distance lower bound together with this strategy. Distance lower bound was never achieved in our testbed.

The walk to root uses vertex locking so the correctness is hold. We need to lineary order these marks (locks) to avoid discovery of non existent negative cycle. We do walk to root and we are marking vertexes with some stamp. If we find marked vertex we decide as follows. Vertex is marked with lesser stamp then we mark this vertex with current stamp. Vertex is marked with the same stamp then we find the negative cycle. Vertex is marked with greater stamp then we stop the walk to root as we do not find the cycle. [BČKP01a]

3.3 The pseudo-codes

One of the processors is called *Manager* that should own the source vertex s and that starts and stops the computing. The variable α identifies the current processor. The function $Owner(u)$ identifies the processor that own vertex u .

Each processor runs the *Main* procedure with source vertex s . The procedure $send_msg$ sends a message to another processor. The procedure $process_messages()$ checks the incoming messages queue and do appropriate action.

The pseudo-codes for D'Escopo Pape and Pallotino heuristics are not included. The modification of Bellman-Ford-More algorithm to those heuristics is trivial.

```
1 proc Main( $s$ )
2   InitializeSingleSource();  $Q^\alpha := empty$ ;
3   if  $\alpha = Manager$  then push( $Q^\alpha, s$ );  $d(s) := 0$ ;  $p(s) := nil$ ; fi
4   while not finished do
5     if  $Q^\alpha \neq empty$  then  $u := pop(Q^\alpha)$ ;  $\{STD, STT, WTR\}\_Scan(u)$ ; fi
6     process\_messages();
7   od

1 proc InitializeSingleSource()
2   foreach  $v \in V$  do if  $Owner(v) = \alpha$  then  $p(v) := nil$ ;  $d(v) := \infty$ ; fi od
```

Bellman–Ford–Moore with Subtree Disassembly

```

1 proc STD_Scan(u)
2   foreach  $v \in Succ(u)$  do
3     if  $Owner(v) = \alpha$ 
4       then STD_Update( $v, u, d(u) + l(\langle u, v \rangle)$ );
5       else send_msg( $Owner(v), "STD\_Update(v, u, d(u) + l(\langle u, v \rangle))"$ ); fi
6   od

```

```

1 proc Update( $v, u, t$ )
2   if  $d(v) > t$ 
3     then  $d(v) := t; p(v) := u;$ 
4     if  $d(u) < threshold$  then "Negative cycle found"; terminate; fi
5      $Std(v, u, l(v, u));$ 
6     if  $v \notin Q^\alpha$  then push( $Q^\alpha, v$ ); fi
7   fi

```

```

1 proc Std( $v, p, l$ )
2   if  $p(v) \neq p$  then return; fi
3   Local  $Q_1$ ; push( $Q_1, (v, l)$ );
4   while  $Q_1$  not empty do
5      $(v_1, l_1) := pop(Q_1);$ 
6     if  $(v_1 = a) \wedge (l_1 < 0)$  then "Negative cycle found"; terminate; fi
7     foreach  $u \in Succ(v_1) \wedge p(u) = v_1$  do
8       if  $Owner(u) = \alpha$ 
9         then push( $Q_1, (u, l_1 + l(\langle v_1, u \rangle))$ );  $p(u) := deleted;$ 
10        if  $u \in Q^\alpha$  then remove( $Q^\alpha, u$ ); fi
11        else send_msg( $Owner(u), "Std(u, p, l_1 + l(\langle v_1, u \rangle))"$ ); fi
12    od od

```

Network Simplex

```

1 proc STT_Scan(u)
2   foreach  $v \in Succ(u)$  do
3     if  $Owner(v) = \alpha$ 
4       then STT_Update( $v, u, l(\langle u, v \rangle), d(u)$ );
5       else send_msg( $Owner(v), "STT\_Update(v, u, l(\langle u, v \rangle), d(u))"$ ); fi
6   od

```

```

1 proc Update( $v, u, luv, t$ )
2   if  $d(v) > t + luv$ 
3     then if  $p(v) = nil$  then  $d(v) := t + luv; p(v) := u;$ 
4     else  $p(v) := u; Pivot(v, u, d(v) - (t + luv), luv);$  fi fi
5   if  $p(v) = u$  then push( $Q^\alpha, v$ ); fi

```

```

1 proc Pivot( $v, u, t, luv$ )
2   LocalQ1; push( $Q_1, (v, luv)$ );
3   while  $Q_1$  not empty do
4      $(v_1, l_1) := pop(Q_1)$ ;
5     if  $(u = v_1) \wedge (l_1 < 0)$  then "Negative cycle found"; terminate; fi
6     if  $(u = v_1)$  then continue; fi
7      $d(v_1) := d(v_1) - t$ ;
8     foreach  $u_1 \in Succ(v_1) \wedge p(u_1) = v_1$  do
9       if Owner( $u_1$ ) =  $\alpha$ 
10        then push( $Q_1, (u_1, l_1 + l(\langle v_1, u_1 \rangle))$ );
11        else send_message(Owner( $u_1$ ), "Pivot( $u_1, u, t, l_1 + l(\langle v_1, u_1 \rangle)$ )"); fi od
12    if  $v_1 \in Q^\alpha$  then remove( $Q^\alpha, v_1$ ); fi od

```

Bellman–Ford–Moore with Walk to Root

```

1 proc WTR_Scan( $v$ )
2   foreach  $(v, u) \in E$  do
3     if Owner( $u$ ) =  $\alpha$ 
4       then WTR_Update( $u, v, d(v) + l(v, u)$ )
5       else send_message(Owner( $u$ ), "WTR_Update( $u, v, d(v) + l(v, u)$ )") fi
6   od

1 proc WTR_Update( $u, v, t$ )
2   if  $d(u) > t$  then if walk( $u$ )  $\neq nil$ 
3     then if Owner( $v$ ) =  $\alpha$ 
4       then push( $Q^\alpha, v$ )
5       else send_message(Owner( $v$ ), "push( $Q, v$ )") fi
6     else  $d(u) := t$ ;  $p(u) := v$ ;
7     if WTR_amortization then WTR( $[u, stamp], u$ );
8      $stamp ++$  fi;
9     if  $u \notin Q^\alpha$  then push( $Q^\alpha, u$ ) fi fi fi

1 proc WTR( $[origin, stamp], at$ )
2   done := false;
3   while  $\neg done$  do
4     if owner( $at$ ) =  $\alpha$ 
5       then if walk( $at$ ) =  $[origin, stamp]$  then "Negative cycle found";
6       terminate; fi
7       if  $(at = source) \vee (walk(at) > [origin, stamp])$ 
8         then if Owner( $origin$ ) =  $\alpha$ 
9           then REM( $[origin, stamp], origin$ )
10          else send_message(Owner( $origin$ ),
11            "REM( $[origin, stamp], origin$ )"); fi
12    done := true; continue; fi

```

```

13         if  $walk(at) = [nil, nil] \vee (walk(at) < [origin, stamp])$ 
14             then  $walk(at) := [origin, stamp]$ ;
15                  $at := p(at)$ ;
16         fi
17     else  $send\_message(Owner(at), "WTR([origin, stamp], at)");$ 
18          $done := true$ ; fi
19 od

1 proc  $REM([origin, stamp], at)$ 
2      $done := false$ ;
3     while  $\neg done$  do
4         if  $Owner(at) = \alpha$ 
5             then if  $walk(at) = [origin, stamp]$  then  $walk(at) := [nil, nil]$ ;
6                  $at := p(at)$ ;
7                 else  $done := true$  fi
8             else  $send\_message(Owner(at), "REM([origin, stamp], at)");$ 
9                  $done := true$  fi
10 od

```

4 Comparison of Distributed Algorithms

5 Conclusions

We provide and analyse distributed algorithms for the general SSSP and NCP problems for graphs specified with the adjacency lists. The algorithms are designed for networks of workstations where the input graph is distributed over individual workstation and workstations communicate via a message passing interface.

Based on our experiments we conclude that in situations where no apriori information about the graph is given the best choice is the Subtree Disassembly algorithm (in the sequential version known also as Tarjan's algorithm). In case of graphs with negative-valued edges the best choice is to co-join this algorithm with the Pallottino heuristic.

References

- [BČKP01a] L. Brim, I. Černá, P. Krčál, and R. Pelánek. Distributed shortest path for directed graphs with negative edge lengths. Technical report, Faculty of Informatics, Masaryk University Brno, 2001.
- [BČKP01b] L. Brim, I. Černá, P. Krčál, and R. Pelánek. Distributed ltl model checking based on negative cycle detection. In *FST TCS 2001: Foundations of Software Technology and Theoretical Computer Science*, number 2245 in Lecture Notes in Computer Science, pages 96–107. Springer-Verlag, 2001.

- [Bel58] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.
- [CG99] B. V. Cherkassky and A. V. Goldberg. Negative-cycle detection algorithms. *Mathematical Programming, Springer-Verlag*, 85:277–311, 1999.
- [CMMS98] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders. A parallelization of Dijkstra’s shortest path algorithm. In *Proc. 23rd MFCS’98*, volume 1450 of *Lecture Notes in Computer Science*, pages 722–731. Springer-Verlag, 1998.
- [Coh96] E. Cohen. Efficient parallel shortest-paths in digraphs with a separator decomposition. *Journal of Algorithms*, 21(2):331–357, 1996.
- [CZ95] S. Chaudhuri and C. D. Zaroliagis. Shortest path queries in digraphs of small treewidth. In *Automata, Languages and Programming*, pages 244–255, 1995.
- [Dan51] G.B. Dantzig. Application of the simplex method to a transportation problem. *Activity Analysis and Production and Allocation*, 1951.
- [DKZ94] P. Spirakis D. Kavvadias, G. Pantziou and C. Zaroliagis. Efficient sequential and parallel algorithms for the negative cycle problem. In *Proc. 5th ISAAC’94*, volume 834 of *Lecture Notes in Computer Science*, pages 270–278. Springer-Verlag, 1994.
- [GR93] A. V. Goldberg and T. Radzik. A heuristic improvement of the Bellman-Ford algorithm. *AMLETS: Applied Mathematics Letters*, 6:3–6, 1993.
- [HTB97] M. Hribar, V. Taylor, and D. Boyce. Performance study of parallel shortest path algorithms: Characteristics of good decompositions. In *Proc. ISUG ’97 Conference*, 1997.
- [HTB98] M. Hribar, V. Taylor, and D. Boyce. Parallel shortest path algorithms: Identifying the factors that affect performance. Technical Report CPDC-TR-9803-015, Center for Parallel and Distributed Computing, Northwestern University, 1998.
- [MS00] U. Meyer and P. Sanders. Parallel shortest path for arbitrary graphs. In *EUROPAR: Parallel Processing, 6th International EURO-PAR Conference*, Lecture Notes in Computer Science. Springer-Verlag, 2000.
- [Pal84] S. Pallottino. Shortest-path methods: Complexity, interrelations and new propositions. *Networks*, 14:257–267, 1984.
- [Pap74] U. Pape. Implementation and efficiency of Moore-algorithms for the shortest path problem. *Mathematical Programming*, 7:212–222, 1974.

- [RV92] K. Ramarao and S. Venkatesan. On finding and updating shortest paths distributively. *Journal of Algorithms*, 13:235–257, 1992.
- [TZ96] J. Traff and C.D. Zaroliagis. A simple parallel algorithm for the single-source shortest path problem on planar digraphs. In *Parallel algorithms for irregularly structured problems (IRREGULAR-3)*, volume 1117 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, 1996.