Detecting negative cycles with Tarjan's breadth-first scanning algorithm

Tibor Ásványi
asvanyi@inf.elte.hu

ELTE Eötvös Loránd University
Faculty of Informatics
Budapest, Hungary

Abstract

The Bellman-Ford algorithm solves the single-source shortest path problem on directed, weighted graphs in the general case: Edges with negative weights are allowed, if there is no negative cycle reachable from the source vertex s. (With such a cycle there is no shortest path to some vertices reachable from s). The algorithm runs in $\Theta(nm)$ time, where n and m are the numbers of vertices and edges respectively. Therefore many improvements have been suggested.

One of them is Tarjan's breadth-first scanning (BFscan) algorithm. The worst-case runtime remains the same, but BFscan performs much better than the Bellman-Ford algorithm in practice.

The original BFscan terminates, *iff* there is no negative cycle reachable from the source vertex. Otherwise the method never halts. In order to make BFscan robust, a few distinct supplements have been proposed. In this paper we introduce a new completion, and compare it with those known to us.

1 Introduction

Given a network represented by a directed graph with possibly negative edge weights, finding shortest paths from a source vertex s to the other vertices is a basic problem [3, 5]. Tarjan's breadth-first scanning algorithm [1, 4] finds such paths to each vertex accessible from s, if each of these optimal paths exist, i.e. there is no negative cycle reachable from s.[1] The algorithm is also known as Queue-based Bellman-Ford [6]. In this paper we introduce a refinement of it. In order to make comparison easier, we closely follow Tarjan's terminology[2], methodology, and order of presentation, but to save space, most of the proofs of his theorems are omitted here.

In the next section we enumerate his results up to his breadth-first scanning algorithm. Then, in section 3, we introduce our refinement which is a new check for negative cycles, and compare it with some previous proposals.

In: E. Vatai (ed.): Proceedings of the 11th Joint Conference on Mathematics and Computer Science, Eger, Hungary, 20th – 22nd of May, 2016, published at http://ceur-ws.org

[1] Thus Tarjan's algorithm solves this route optimization problem in the most general case. In addition, there are interesting applications of finding negative cycle(s) in a graph. See for example the *arbitrage* problem (24-3 in [5]).

[2] For example, Tarjan's (and so our) definition of the path is identical with the usual notion of walk, i.e. a path may contain repeated vertices, like in [1, 2, 4, 5, 6].

2 Tarjan's breadth-first scanning algorithm

In this section we introduce first the basic notions, and the formal problem. Next we give an initial solution, and then refine it, following Tarjan [1].

2.1 The problem

Let $G = (V, E)$ be a directed graph, where $|V| = n > 0$, $E \subseteq V \times V$,
$|E| = m \in 0..n^2$, $length : E \to \mathbb{R}$ (possibly negative), $s \in V$.

Definition 2.1. $p =< v_0, \ldots, v_k >$ $(k \geq 0)$ is a path[2], *iff* $\{v_0, \ldots, v_k\} \subseteq V$, and $\{(v_{i-i}, v_i) | i \in 1..k\} \subseteq E$. $length(p) = \sum_{i=1}^{k} length(v_{i-i}, v_i)$, $e(p) = k$.
Path p is a cycle, *iff* $k > 0 \wedge v_0 = v_k$. Cycle p is negative, *iff* $length(p) < 0$.

Definition 2.2. Let $s, t \in V$; t is reachable form s, *iff* there is some path from s to t, i.e. there is a $< v_0, \ldots, v_k >$ $(k \geq 0)$ path so that $v_0 = s \wedge v_k = t$

Problem 2.3. *An important network optimization problem is the* single source problem: *Given a source $s \in V$, find a shortest path from s to v for each $v \in V$ reachable from s.*

Any other shortest path finding problem can be reduced to this one [1], so we consider this here. A compact way to represent a solution is the *shortest-path tree* rooted at s each of whose paths is a shortest path in G [1]. There is a well-known condition on the existence of shortest (i.e. optimal) path [1]:

Theorem 2.4. *Let $s, t \in V$ such that t is reachable form s. There is a shortest path form s to t,* iff *no path from s to t contains a* negative cycle.

In this paper we develop an algorithm which computes a shortest-path tree or detects a negative cycle reachable from s. In the latter case, clearly there is no solution.

If the algorithm computes a shortest-path tree, then for each reachable vertex $v \neq s$, label $p(v)$ shows its parent in the tree and $d(v)$ shows the distance (i.e. the length of the optimal path from s to v); $p(s) = NIL$, $d(s) = 0$; and for any non-reachable vertex u, $p(u) = NIL$, $d(u) = \infty$.

2.2 Ford's labeling method

This algorithm initializes each vertex as if non-reachable, except that $d(s) = 0$. Later it maintains a tree rooted at s which approximates step-by-step the result. After the previous initialization, it repeats the *labeling step* while it is applicable:

LABELING STEP (Ford).
Select an edge (v, w) such that $d(v) + length(v, w) < d(w)$.
Replace $d(w)$ by $d(v) + length(v, w)$ and $p(w)$ by v.

Now we enumerate the most important properties of *Ford's labeling method* (1956).

Lemma 2.5. *The labeling method maintains the invariant that if $d(v) < \infty$, there is a path form s to v of length $d(v)$.*
(Proof: by induction on the number of labeling steps.)

Theorem 2.6. *When the labeling method halts, $d(v)$ is the length of a shortest path from s to v, if v is reachable from s, and $d(v) = \infty$ otherwise. If there is a negative cycle reachable from s, the method never halts.*
(Proof: Let p be some path from s to v. When the labeling method halts, $length(p) \geq d(v)$ by induction on $e(p)$ (see Definition 2.1). Now with Lemma 2.5, $d(v)$ is the length of a shortest path from s to v. If v is not reachable from s, $d(v) = \infty$ with the same lemma. If there is a negative cycle reachable from s, consider some element t of this cycle. Now – with Theorem 2.4 – there is no shortest path from s to t, but if the method halted, $d(t)$ would be the length of a shortest path from s to t, which proves that the method never halts.)

Lemma 2.7. *If at some time during the labeling method $p^k(v) = v$ for some $v \in V$ and k positive integer, then the corresponding cycle in G is negative.*

The previous lemma helps us to identify negative cycles [1]. The next theorem makes sure, that when the labeling method halts, we can find a shortest-path tree besides the appropriate d values of the accessible vertices [1].

Theorem 2.8. *When the labeling method halts, the parent pointers define a shortest-path tree for the subgraph of G induced by the vertices reachable from s.*

If no negative cycle is reachable from s, Ford's method halts, but it may be exponential in m [1]. We need refinements.

Our algorithm, and each known algorithm which solves the single source problem (Problem 2.3) is a special case of Ford's labeling method, except that they may add various halting conditions in order to handle the case when this algorithm goes into infinite loop (i.e. there is a negative cycle reachable from s).

2.3 The labeling and scanning method

Tarjan's first refinement is the *labeling and scanning method* [1]. We maintain a partition of the vertices into three states: *unlabeled*, *labeled*, and *scanned*.

With the usual initialization, s becomes labeled, and any other vertex unlabeled. Then we repeat the scanning step while there is any labeled vertex.

SCANNING STEP: Select a labeled vertex v and *scan* it, thereby converting it to the scanned state, by applying the labeling step to each edge (v, w) such that
$d(v) + length(v, w) < d(w)$, thereby converting w to the labeled state.

The following invariant [1] of the *labeling and scanning method* ensures that the labeling steps can be restricted to the edges coming out from the labeled vertices.

Lemma 2.9. *In the* labeling and scanning method*, if $d(v) + length(v, w) < d(w)$, then v is in the labeled state.*

2.4 The breadth-first scanning method

Efficient scanning orders (like topological scanning, and Dijkstra's algorithm [5]) can be defined for special graphs, while in the general case a good scanning order is *breadth-first* [1, 2]. To this end, in breadth-first scanning (in BFscan, and later in CBFscan) we store the labeled vertices in queue Q.

```
procedure BFscan( vertices V, edges E, vertex s )
  for each u in V
      d(u):= INFTY; p(u):= NIL; state(u):= unlabeled
  d(s) := 0;
  queue Q := [s]; state(s) := labeled;
  while( Q \= [] )
      u := deQueue(Q); state(u) := scanned;
      for each v : (u,v) in E
          if d(u)+length(u,v) < d(v)
              d(v) := d(u)+length(u,v);
              p(v) := u;
              if state(v) \= labeled
                  enQueue(Q,v); state(v) := labeled
```

Because procedure BFscan is a special case of the labeling method, Theorem 2.6 and 2.8 imply that, if BFscan halts, then it computes the shortest-path tree with the appropriate $d(v)$ distance values and $p(v)$ parent pointers. Form Theorem 2.6 we know that if there is a negative cycle reachable from s, BFscan never halts. The following theorem makes sure that otherwise it stops in $O(nm)$ time. First we divide its execution into passes [1].

Definition 2.10. In procedure BFscan, pass zero consists of the initial scanning of the source s. For $j > 0$, pass j consists of the next scanning of all the vertices on the queue at the end of pass $j - 1$.

Theorem 2.11. *If there is no negative cycle reachable form s, then BFscan runs in $O(nm)$ time, stopping by the end of pass $n-1$. Otherwise BFscan never halts.*

In order to make BFscan robust, one must ensure that it halts even if there is a negative cycle reachable form s. Tarjan [1, 2] proposed to count passes: If the queue becomes empty by the end of pass $n-1$, then there is no negative cycle reachable form s. Otherwise at the end of pass $n-1$ we can select any vertex from the queue, and starting from it, we can follow the parent pointers. Among the ancestors we find a cycle, and because of Lemma 2.7, it is a negative cycle reachable from s.

3 Our check for negative cycles

Now we give another, stronger condition based on our new $e(v)$ attribute of the labeled and scanned vertices. With this condition, our algorithm never stops later, and sometimes earlier then Tarjan's pass counting BFscan.

We introduce our new label $e(v)$ of the vertices into BFscan. It counts the number of edges on the shortest (i.e. lightest or cheapest) path found to vertex v (see Definition 2.1). We prove, that if $e(v) \geq n$, then among the ancestors of v we can find a cycle, and this is a negative cycle reachable from s. And if there is a negative cycle reachable from s, then we find $e(v) \geq n$ before the end of pass $n-1$. Based on this, we extend BFscan with the necessary check, as it is shown in Figure 1. Our CBFscan function returns NIL, if it computes the shortest-path tree. If it recognizes that for some $v \in V$, $e(v) \geq n$, it returns v. Then it is easy to find a negative cycle among the ancestors of v in $\Theta(n)$ time. (See Figure 2.)

```
vertex function CBFscan( vertices V, edges E, vertex s )
 1:  for each u in V
 2:      d(u) := INFTY;  p(u) := NIL;  state(u) := unlabeled
 3:  d(s) := 0;  e(s) := 0
 4:  queue Q := [s];  state(s) := labeled
 5:  while( Q \= [] )
 6:      u := deQueue(Q);  state(u) := scanned
 7:      for each v : (u,v) in E
 8:          if( d(u)+length(u,v) < d(v) )
 9:              d(v) := d(u)+length(u,v)
10:              p(v) := u;  e(v) := e(u)+1
11:              if( e(v) < n )
12:                  if state(v) \= labeled
13:                      enQueue(Q,v);  state(v) := labeled
14:              else return v  // Neg.cycle among ancestors of v
15:  return NIL  // Shortest-path tree computed
```

Figure 1: Checked Breadth-First scanning

```
vertex function FindNegCycle( vertices V, vertex v )
  booelan B[V];  for each u in V,  B[u] := false
  B[v] := true;  vertex u := p(v)
  while( not B[u] )  B[u] := true;  u := p(u)
  return u  // u is a vertex of a negative cycle
```

Figure 2: Find a vertex of a Negative Cycle among the ancestors of v

Now we state and prove the necessary properties of CBFscan. Let I_5, and I_{11} denote two invariants of the program at the beginning of the appropriate lines.

Lemma 3.1. *Let $P(w)$ be the following assertion: the path $q(w) = [p^{e(w)}(w), p^{e(w)-1}(w), \ldots, p(w), w]$ exists and $(\forall x \in q(w))(d(x) < \infty$ and x is reachable from $s)$. Thus the while loop of function CBFscan at line 5 maintains the following invariant I_5: for each vertex w, if state(w) = labeled then $n > e(w) \geq 0 \wedge P(w)$. Also at line 11 we have the invariant I_{11}: $I_5 \wedge n > e(u) \geq 0 \wedge P(u) \wedge n \geq e(v) \geq 0 \wedge P(v)$.*

Proof. First I_5 is clearly true because only s is labeled, $Q = [s]$ (in general Q contains the labeled vertices), and $e(s) = 0 \wedge d(s) = 0 \wedge q(s) = [s]$.

Supposing that at line 5 I_5 is true, there are too cases. (I) If From line 5 the program goes directly to line 11, I_{11} will also be true. Only the existence of $q(v)$ is nontrivial: If $v \notin q(u)$ then $q(v) = q(u)$ concatenated with $[v]$. If $v \in q(u)$ then for some $k \in 1..e(v), v = p^k(v)$; so we have found a cycle, and $q(v)$ exists. (II) If From line 5 the program goes to line 5 without touching line 11, I_5 remains clearly true for the remaining labelled vertices.

Provided that at line 11 $I_{11} \wedge e(v) < n$, there are two cases again. (1) From line 11 we go to line 11 without touching line 5, and with the new Q and v I_{11} remains true. (2) From line 11 we go to line 5 without touching line 11, and with the new Q I_5 will be true.

Consequently both of I_5 and I_{11} are invariants. □

Theorem 3.2. *If in the test in line 11, $e(v) \geq n$ is found, then we can find a cycle following the parent pointers of v; this cycle is negative, and reachable form s.*

Proof. $I_{11} \wedge e(v) \geq n$ implies $e(v) = n$. Thus the number of the nodes on the path $q(v)$ is $n + 1$, and so $(\exists i, j \in 0..n)(i > j \wedge p^i(v) = p^j(v))$. Lemma 2.7 and I_{11} imply that the cycle $[p^i(v), \ldots, p^j(v))]$ is negative, and reachable form s. □

Lemma 3.3. *If CBFscan is in pass j, for each labeled vertex u (to be) scanned in this pass, $e(u) \geq j$, and for each labeled vertex v to be scanned in the next pass, $e(v) > j$.*

Proof. Only s is scanned in pass 0, and $e(s) = 0$. For each successor v of s, $e(v) = 1$ and v is scanned in pass 1. Then the lemma comes by induction on the number of passes: Let us suppose that we are in pass j, and the invariant of this lemma stands. The new $e(v)$ values are generated in line 10, with the statement $e(v) := e(u)+1$, where u is a vertex being scanned in pass j, and so $e(u) \geq j$. Thus $e(v) \geq j + 1$, and this fits the invariant, regardless of whether v is a vertex to be scanned yet in pass j or only in pass $j + 1$. □

Theorem 3.4. *If no negative cycle is reachable form s, the run of CBFscan follows the run of BFscan. If a negative cycle is reachable from s, then CBFscan at line 11 finds a vertex v with $e(v) = n$. In any case, CBFscan halts at the latest during pass $n - 1$, and runs in $O(nm)$ time.*

Proof. If there is no negative cycle reachable form s, Theorem 3.2 implies that the test in line 11 always finds $e(v) < n$, the run of CBFscan follows the run of BFscan, and with Theorem 2.11 CBFscan also stops by the end of pass $n - 1$.

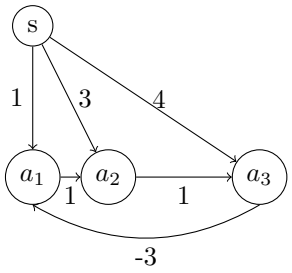
If a negative cycle is reachable from s, Theorem 2.11 makes sure that BFscan performs its pass $n - 1$, n, and so on, infinitely. While CBFscan in line 11 finds $e(v) < n$, the run of CBFscan follows the run of BFscan. This means that if CBFscan halts, it halts because at the test in line 11 $e(v) \geq n$, and considering I_{11} in Lemma 3.1, it stops with $e(v) = n$. If CBFscan does not halt by the end of its first $n - 2$ pass, it halts during its pass $n - 1$: Considering Lemma 3.3 we have that each labeled vertex u being scanned in this pass has $e(u) \geq n - 1$. We can suppose that during pass $n - 1$ CBFscan goes at least once to the test in line 11. (Otherwise at the end of pass $n - 1$ CBFscan would find $Q = []$ at line 5, and it would run and stop following BFscan, but BFscan does not halt in this case.) At line 11, $e(u) \geq n - 1$. Consequently $e(v) \geq n$. Then CBFscan halts.

Regardless of whether a negative cycle is reachable from s or not, CBFScan stops by the end of pass $n - 1$. Thus its total runtime is $O(nm)$. (Each pass requires $O(m)$ time, since in a pass maximum m edges are processed, in pass 0 only s is scanned, and from pass 1, during a pass (i) each vertex is scanned at most once and (ii) not more than $min(n, m)$ vertices are scanned [1].) □

3.1 Comparing our check for negative cycles and previous proposals

If there is no negative cycle reachable from s, we have seen that both of Tarjan's pass counting BFscan, and our CBFscan follows the run of BFscan, and they stop at the same point with $Q = []$. If a negative cycle is reachable from s, Tarjan's pass counting BFscan stops at the end of pass $n - 1$, while our CBFscan halts *at the latest during* pass $n - 1$. However, the reader can check, that for example on the graph shown at Figure 3, CBFscan halts by the end of the first pass.

And it is easy to create similar graphs of any size so that CBFscan recognizes the negative cycle, and halts by the end of the first pass (e.g. let the vertices be $\{a_0, a_1, a_2, \ldots a_n\}$ where $s = a_0$, $length(a_{i-1}, a_i) = 0$ $(i = 1..n)$, $length(a_0, a_j) = 1$ $(j = 2..n)$), and $length(a_n, a_k) = -1$ for some $k \in [0..(n - 1)]$).



This graph contains 4 vertices, so procedure CBFscan is guaranteed to halt by the end of pass 3, but – supposing that the vertices at a scanning step are put into Q in the order of the indexes – it recognizes a negative cycle and stops while scanning the last vertex of pass 1.

Figure 3: For example on this graph CBFscan halts earlier

Weiss (1995) gives a weaker condition: "by stopping the algorithm[3] after any vertex has dequeued $|V|+1$ times[4], we can guarantee termination."[4] This version finds a negative cycle only in pass n or later, because a vertex is dequeued at most once in a pass. Here we suggest a trivial improvement of the previous condition: *The robust BFscan should stop if any vertex v is enqueued n times, because it means that graph contains a negative cycle.* (Let us call BFscan with this improved termination condition *WiBFscan.*) If there is no reachable negative cycle, WiBFscan is as fast as our CBFscan, or the pass counting BFscan of Tarjan. Otherwise it can halt sooner or later than Tarjan's pass counting BFscan, but not sooner than our algorithm:

When any vertex v is enqueued nth time, we are at least in pass $n-1$, because a vertex is enqueued at most once in a pass. In addition, this v will be still in the queue at the end of pass $n-1$, and Theorem 2.11 implies that there is a reachable negative cycle. Thus it seems to be possible that WiBFscan finds the negative cycle earlier than the pass counting BFscan of Tarjan. But a vertex may not be enqueued in each pass.[5] So this algorithm may find the negative cycle later than Tarjan's version. Surely WiBFscan does not find the negative cycle earlier than our CBFscan, because in our algorithm, provided that vertex v is labeled during pass $n-1$, $e(v) \geq n$, and our algorithm halts, if not earlier.

Let us note, that in the implementation of WiBFscan, for each vertex v we can use a counter of enqueuing events $en(v)$. We suggest that one should initialize $en(s)$ to $n-1$ immediately before the main loop of WiBFscan, because the first enqueuing of s in the main loop already shows a negative cycle. Thus the early recognition of a negative cycle going through the start vertex becomes possible without extra cost (although this is a only a special case).

3.2 An open question

Tarjan proposed another, more complex robust version of BFscan which stops "as soon as the parent pointers define a cycle" [1]: "when processing an edge (u, v) such that $d(u) + length(u, v) < d(v)$", "we can look for u among the descendants of v". This "method requires storing extra information about the tree (a list of the vertices in preorder will do) but if carefully implemented preserves the $O(nm)$ running time".

It is clear that if there is no reachable negative cycle, in practice this complex robust BFscan is slower then the other (robust) versions of the BFscan program considered here, because of its complex check.

If there is reachable negative cycle, it usually performs less number of scanning steps than the previously examined robust versions of BFscan because it stops "as soon as the parent pointers define a cycle", but still there is no comparison between the practical runtime of this complex robust BFscan and that of CBFscan.

4 Conclusions

If there is no negative cycle reachable from s, the runtime of our CBFscan program, and that of the other robust BFscan versions detailed here are practically the same. In this case Tarjan's more complex robust algorithm (considered in Subsection 3.2) is slower than the other programs.

Otherwise CBFscan finds a negative cycle at last during pass $n-1$. Tarjan's pass counting BFscan finds it only after pass $n-1$. Weiss's original version is the weakest because it finds it only in the nth pass or later. Our

[3] BFscan

[4] $n+1$ times

[5] We believe, the vertices probably are not enqueued in each pass.

improved version WiBFscan finds it sooner or later than Tarjan's pass counting BFscan, but not sooner than our CBFscan (except that with our trick a negative cycle through s is found as soon as in BFscan it is possible).

There are cases, when CBFscan finds a negative cycle reachable from s even before pass $n-1$, and it is much faster than the other simple versions of robust BFscan.

If there is a negative cycle reachable form s, Tarjan's more complex robust algorithm (see Subsection 3.2) performs the minimal number of scanning steps, but still we have no comparison of the practical runtime of this algorithm and that of CBFscan in this case.

References

[1] Tarjan, Robert Endre, Data Structures and Network Algorithms, *CBMS-NSF Regional Conference Series in Applied Mathematics*, 1987.

[2] Tarjan, Robert Endre, A unified approach to path problems, *J. Assoc. Comput. Mach., 28, pp. 577-593*, 1981.

[3] Bellman, R.E., On a Routing Problem, *Quarterly of Applied Mathematics, 16 (1958), 87-90.*

[4] Weiss, M.A., Data Structures and Algorithm Analysis, *Addison-Wesley*, 1995, 1997, 2007, 2012, 2013.

[5] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., Introduction to Algorithms, *The MIT Press*, 2009.

[6] Sedgewick, R., Wayne, K., Algorithms, 4th Edition *Addison-Wesley Professional*, 2011. ISBN 0-321-57351-X (Ebook: http://algs4.cs.princeton.edu/home/)