Negative-Weight Cycle Algorithms

Xiuzhen Huang
Department of Computer Science,
Arkansas State University,
State University, Arkansas 72467 USA.
xzhuang@csm.astate.edu

Abstract

The problem of finding a negative cycle in a weighted, directed graph is discussed here. First the algorithm for printing out a negative cycle reachable from the source s, with the running time no worse than the Bellman-Ford algorithm for the single-source shortest-path problem, is presented. Then an approach with the same time complexity, which could be used for outputting a negative cycle that may not be reachable from s, is reported.

Keywords: algorithm, Bellman-Ford algorithm, graph, negative-weight cycle

1 Negative-weight Cycle Problem

The problem of finding a cycle of negative-weight in a weighted, directed graph is a classic problem in algorithm design and analysis. This problem "comes up both directly, for example in currency arbitrage, and as a sub-problem in algorithms for other graph (or, network) problems, for example the minimum-cost flow problem [2]."

Given a weighted, directed graph $G = (V, E)$, the single-source shortest-path problem is to find the shortest paths from a specific source vertex s to every other vertex of the graph G. Dijkstra's algorithm solves this problem if all edge weights are nonnegative values. The Bellman-Ford algorithm solves the single-source shortest-path problem in general case where edge weights could be negative values.

As is stated in [3], if a graph $G = (V, E)$ contains no negative-weight cycles reachable from the source vertex s, then for each vertex $v \in V$, the weight of the shortest-path from s to v is well defined, even if the weight of the shortest path might have a negative value. If there is a negative-weight cycle reachable from s, then the weights of the shortest-paths from s to other vertices are not well-defined.

In this paper, the algorithms for finding negative-weight cycles are derived from the Bellman-Ford algorithm for the single-source shortest-path problem. (This is an exercise problem in [3].) Our goal here is to output a negative-weight cycle if such a cycle exists in the given weighted, directed graph, such that the running time is no worse than that of the Bellman-Ford algorithm. The main ideas of the approaches discussed in this report are based on the analysis in [1]-[3].

2 Basic Definitions and Algorithmic Preparations

Readers are referred to [3] for the following definitions. Let $G = (V, E)$ be a weighted, directed graph, with weight function W such that for each edge (u, v), the edge weight $W(u, v)$ is a real number. The length $l(p)$ of a path $p =< v_0, v_1, ..., v_k >$ is the sum of the weights of its constituent edges. The weight of the shortest path from vertex u to v, (u, v), is the minimum $l(p)$, where p is a path from u to v. If there is no path from u to v, $l(p)$ is defined as infinity.

Given a weighted, directed graph $G = (V, E)$ and a source vertex $s \in V$. A shortest-path tree rooted at the source vertex s is a directed subgraph $G' = (V', E')$, where V' and E' are subsets of V and E respectively, such that

1. V' is the set of vertices reachable from s

in G;

2. G' forms a rooted tree with root s, and

3. For all $v \in V'$, the unique path from s to v in G' is the shortest path from s to v in G.

The process of relaxing an edge (u, v) consists of testing whether we can improve the shortest path to vertex v found so far by going through vertex u. If so, we update $d(v)$ and $\pi(v)$, where $d(v)$ initialized as infinity maintains an upper bound on the length of a shortest path from the source vertex s to v, $\pi(v)$ maintains the predecessor of the vertex v, which may be another vertex or null. Please refer to the following pseudo code from [3]:

INITIALIZE-SINGLE-SOURCE (G, s)

Step 1. for each vertex $v \in V$ of G, do

Step 2. $\quad d(v) = \infty$;

Step 3. $\quad \pi(v)$ = null;

Step 4. $d(s) = 0$;

RELAX (u, v, W)

Step 1. if $d(v) > d(u) + W(u, v)$

Step 2. $\quad$ then $d(v) = d(u) + W(u, v)$;

Step 3. $\quad\quad \pi(v) = u$;

To prove the correctness of the shortest paths algorithms, there are several properties of shortest paths and relaxations as follows (see [3]).

Subpaths of the shortest paths are shortest pathes

Given a weighted, directed graph $G = (V, E)$ with weight function $W : E \rightarrow R$, let $p =< v_1, v_2, ..., v_k >$ is a shortest path from v_1 to v_k and, for any i and j such that $1 \leq i \leq j \leq k$, let $p_{ij} =< v_i, v_{i+1}, ..., v_j >$ be the subpath of p from vertex v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.

Triangle inequality

For any edge $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + W(u, v)$.

Upper-bound property

We always have $d(v) \geq \delta(s, v)$ for all vertices $v \in V$, and once $d(v)$ achieves the value $\delta(s, v)$, it never changes.

No-path property

if there is no path from s to v, then we always have $d(v) = \delta(s, v) = \infty$.

Convergence property

If s to $u \rightarrow v$ is a shortest path in G for some $u, v \in V$, and it $d(u) = \delta(s, u)$ at ant time prior to relaxing edge (u, v), then $d(v) = \delta(s, v)$ at all times afterward.

Path-relaxation property

If $p =< v_0, v_1, ..., v_k >$ is a shortest path from $s = v_0$ to v_k, and the edges of p are relaxed in the order $(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)$, then $d(v_k) = \delta(s, v_k)$. This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of p.

Predecessor-subgraph property

Once $d(v) = \delta(s, v)$ for all $v \in V$, the predecessor subgraph is a shortest-paths tree rooted at s.

As stated in [3], “Some shortest-paths algorithms, such as Dijkstra’s algorithm, assume that all edges weights in the input graph are nonnegative, ... Others, such as the Bellman-Ford algorithm, allow negative-weight edges in the input graph and produce a correct answer as long as no negative-weight cycles are reachable from the source. Typically, if there is such a negative-weight cycle, the algorithm can detect and report its existence”.

The Bellman-Ford algorithm initializes the distance from one vertex $v \in V$ to the source vertex s, $d(v)$, to be 0 and to all other vertices to infinity. It then does $(|V| - 1)$ passes of relaxation over all edges. It progressively decreases $d(v)$, which could be considered as the estimate on the weight of the shortest path from the source vertex s to the vertex v. After that it checks each edge again to detect negative-weight cycles. It returns FALSE if there is a negative-weight cycle reachable from the source vertex s. Otherwise, after the $(|V| - 1)$ passes of relaxation, $d(v)$ is equal to the weight of the shortest-path from s to v, $\delta(u, v)$. Refer to the following pseudo code from [3].

BELLMAN-FORD (G, W, s)

Step 1. INITIALIZE-SINGLE-SOURCE (G, s)

Step 2. for $i = 1$ to $|V| - 1$

Step 3. do for each edge $(u, v) \in E$

Step 4. do RELAX (u, v, W)

Step 5. for each edge $(u, v) \in E$

Step 6. do if $d(v) > d(u) + W(u, v)$

Step 7. return FALSE

Step 8. return TRUE

The time complexity of the Bellman-Ford algorithm is $O(|V||E|)$, where $|V|$ is the number of vertices and $|E|$ is the number of edges of the graph. This time complexity is the best time bound for the single source shortest path problem [2].

for proving the correctness of the Bellman-Ford algorithm, the following lemma was showed in [3].

Lemma 2.1 *Let $G = (V, E)$ be a weighted, directed graph with source s and weight function $W : E \to R$, and assume that G contains no negative-weight cycles that are reachable from s. Then, after the $|V| - 1$ iterations of the for loop of step 2-4 of the Bellman-Ford algorithm, we have $d(v) = \delta(s, v)$ for all vertices that are reachable from s.*

PROOF. The lemma is proved by appealing to the path-relaxation property. Consider any vertex v that is reachable from s, and let $p =< v_0, v_1, ..., v_k >$, where $v_0 = s$ and $v_k = v$, be any acyclic shortest path from s to v. Path p has at most $|V| - 1$ edges, and so $k \leq |V| - 1$. Each of the $|V| - 1$ iterations of the for loop of step 2-4 relaxes all $|E|$ edges. Among the edges relaxed in the ith iteration, for $i = 1, 2, ..., k$, is (v_{i-1}, v_i). By the path-relaxation property, therefore $d(v) = d(v_k) = \delta(s, v_k) = \delta(s, v)$.

□

Corollary 2.2 *Let $G = (V, E)$ be a weighted, directed graph with source s and weight function $W : E \to R$. Then for each vertex $v \in V$, there is a path from s to v if and only if the Bellman-Ford algorithm terminates with $d(v) < \infty$ when it runs on G.*

Theorem 2.3 *(Correctness of the Bellman-Ford algorithm, [3]) Let Bellman-Ford algorithm be run on a weighted, directed graph $G = (V, E)$ with source s and weight function $W : E \to R$. If G contains no negative-weight cycles that are reachable from s, then the algorithm returns TRUE, we have $d(v) = \delta(s, v)$ for all vertices $v \in V$, and the predecessor graph G_π is a shortest-paths tree rooted at s. If G does contain a negative-weight cycle reachable from s, then the algorithm returns FALSE.*

A sketch of the proof of this result is provided in the next section.

3 Negative-weight Cycle Reachable From the Source Vertex

In this section, we first give the algorithm that will find a negative-weight cycle reachable from the source vertex s in the given weighted, directed graph $G = (V, E)$, if such a negative-weight cycle exists in G. A similar algorithm is presented in [4]. Then we analysis the time complexity of the algorithm and prove its correctness.

As is pointed out in [2] that "all known algorithms for the negative-weight cycle problem combine a shortest path algorithm and a cycle detection strategy". The cycle detection strategy we use here is based on the fact that if the distance label of a vertex v, $d(v)$, is smaller than the length of a shortest simple path from s to v, then the input graph has a negative-weight cycle.

Algorithm A - Finding A Negative-Weight Cycle.

Input: A weighted, directed graph $G = (V, E)$ with edge weight $W(u, v)$ being a real number for each edge (u, v), and a source vertex s.

Output: a negative-weight cycle reachable from the source vertex s if such a cycle exists in graph G; Otherwise, output the information that no negative-weight cycles reachable from the source vertex s.

Step 1. Initialize and execute $(|V| - 1)$ passes of relaxation as in the Bellman-Ford algorithm.

Step 2. Check if there is an edge (u, v) such that $d(u) + W(u, v) < d(v)$. If not, return "no negative-weight cycles reachable from the source vertex s".

Step 3. Otherwise, go backward from v along the predecessor chain, until a cycle is found, i.e., until either v is reached, or some vertex was reached twice. Output the cycle.

We analysis the time complexity of the Algorithm A: Step 1 takes time $O(|V||E|)$ as the Bellman-Ford algorithm; Step 2 checks each edge of the graph, taking time $O(|E|)$; Step 3 needs time bounded by $O(|E|)$. Therefore, the overall time of the algorithm is bounded by $O(|V||E|)$, which is the same as that of the Bellman-Ford algorithm. In the following we prove:

Theorem 3.1 *Algorithm A for finding a negative-weight cycle reachable from a source vertex in a given graph is correct.*

First we show that the Bellman-Ford algorithm is correct and that an edge (u, v) will be found in Step 2 if and only if G has a negative-weight cycle.

Lemma 3.2 *([3]). Let $G = (V, E)$ be a weighted, directed graph with the source vertex s. If graph G contains no negative-weight cycles that are researchable from s, then after $(|V| - 1)$ passes of the relaxations of the Bellman-Ford algorithm, we have that for each vertex v, $d(v)$ is equal to the length of a shortest simple path from the source vertex s to the vertex v, $\delta(s, v)$, for all v that are researchable from the source vertex s. If there is a negative-weight cycle in graph G, then the Bellman-Ford algorithm returns FALSE.*

PROOF. The following proof is adapted from those in [3], [5].

Case 1: Graph $G = (V, E)$ doesn't contain any negative-weight cycles reachable from the source vertex s.

This case can be proved by induction: if there is a shortest simple path p from s to v containing k edges, then after k passes of relaxations $d(v) = l(p)$, where $l(p)$ is the length of the path p.

Consider a shortest path $p =< v_0, v_1, ..., v_k >$, where $v_0 = s$ and $v_k = v$, be any acyclic shortest path from the source vertex s to the vertex v. Path p has at most $(|V| - 1)$ edges, therefore we have $k \le (|V| - 1)$.

Proof by induction:

$d(s) = 0$ after initialization;

Assume $d(v_{i-1})$ is a shortest path after iteration $(i - 1)$;

Since edge (v_{i-1}, v_i) is updated on the ith pass, $d(v_i)$ must then reflect the shortest path to v_i;

Since we perform $(|V| - 1)$ iterations, $d(v_i)$ for all reachable vertices v_i must now represent shortest paths, that is $d(v_i) = (s, v_i)$.

If graph G contains no negative-weight cycles that are researchable from s, the algorithm will return TRUE because on the $|V|$th iteration, no distances will change.

Case 2: Graph $G = (V, E)$ contains a negative-weight cycle $c =< v_0, v_1, ..., v_k >$, where $v_0 = v_k$, reachable from the source vertex s.

Proof by contradiction: The cycle $c =< v_0, v_1, ..., v_k >$ is a negative-cycle reachable from s, then we have

$$\sum_{i=1}^{k} W(v_{i-1}, v_i) < 0. \quad (1)$$

Assume the Bellman-Ford algorithm returns TRUE. Thus,

$$d(v_{i-1}) + W(v_{i-1}, v_i) \geq d(v_i), \quad (2)$$

for $i = 1, ..., k$.

Summing the inequalities around the cycle $c =< v_0, v_1, ..., v_k >$ gives us:

$$\sum_{i=1}^{k} d(v_{i-1}) + \sum_{i=1}^{k} W(v_{i-1}, v_i) \geq \sum_{i=1}^{k} d(v_i) \quad (3)$$

Since $v_0 = v_k$, each vertex in the cycle c appears exactly once in each of the summations $\sum_{i=1}^{k} d(v_{i-1})$ and $\sum_{i=1}^{k} d(v_i)$, so

$$\sum_{i=1}^{k} d(v_{i-1}) = \sum_{i=1}^{k} d(v_i) \quad (4)$$

Moreover, $d(v_i)$ is finite for $i = 1, ..., k$. Therefore,

$$\sum_{i=1}^{k} W(v_{i-1}, v_i) \geq 0. \quad (5)$$

This leads to a contradiction. We conclude that the Bellman-Ford algorithm returns TRUE if the graph G contains no negative-weight cycle reachable from the source vertex s, and FALSE otherwise. □

To prove the theorem of the correctness of Algorithm A, we need the following lemmas and corollaries from [2].

Define the predecessor graph G_π of the Bellman-Ford algorithm is the subgraph induced by the edge $(\pi(v), v)$ for all v where $\pi(v) \neq$ null.

Corollary 3.3 *If the predecessor graph G_π is acyclic, then it is a tree rooted at s. (It is called the shortest-paths tree.)*

Note that this corollary can be proved by showing the following statement that "let $G = (V, E)$ be a weighted, directed graph with weight function W, let s be the source vertex, and assume that G contains no negative-weight cycles that are reachable from s. Then after the graph is initialized by INITIALIZE-SINGLE-SCOUCE (G, s), the predecessor subgraph G_π forms a rooted tree with root s, and any sequence of relaxation steps on edges of G maintains this property as an invariant [3]".

Corollary 3.4 *Any cycle in the predecessor graph G_π is a negative-weight cycle.*

Corollary 3.5 *If $d(s) < 0$, then the predecessor graph G_π has a negative-weight cycle.*

Lemma 3.6 *Suppose for some vertex v, d(v) is less than the length of a shortest simple path from s to v, then the predecessor graph G contains a cycle c and $w(c) < 0$. (Since d(v) is non-increase, G_π has a cycle at any later point of the execution.)*

PROOF. ([2]) Note that the parent of a vertex has a finite distance and all vertices with finite distances except s has parents. The source vertex s has a parent if and only is $d(s) < 0$.

Suppose we start at v and follow the parent pointers. If we find a cycle of parent pointers in this process, we are done. The only way we can stop without finding a cycle is if we reach s and $d(s) = 0$. In this case there is a simple s-to-v path p in G. From the fact that there is no negative-weight cycle and $d(s) = 0$, we have $d(v) \geq l(p)$. This is a contradiction. □

Lemma 3.7 *If G contains a negative-weight cycle researchable from the source vertex s, then after the relaxation operation of pass $|V|$, G_π always contains a negative-weight cycle.*

PROOF. ([2]) From Lemma 1, we know that after $(|V| - 1)$ iterations of the relaxation, we have that for each vertex v, $d(v)$ is at least as small as the length of a shortest simple path from the source vertex s to the vertex v, $\delta(s, v)$, for all v that are researchable from the source vertex s. The first relaxation after that reduces a distance $d(v)$ below the shortest simple path length $\delta(s, v)$. Therefore, from Lemma 2, we know that the predecessor graph G_π contains a negative-weight cycle. This completes the proof of the lemma. □

4 Negative-weight Cycle that May Not Be Reachable From the Source Vertex

Given a weighted, directed graph $G = (V, E)$, with weight function W such that every edge weight $W(u, v)$ is a real number, for each edge (u, v), and a source vertex s, the Algorithm A discussed in the last section finds a negative-weight cycle that is reachable from the source vertex s.

We present the following approach proposed in [1], which could find a negative-weight cycle in the given weighted, directed graph G that may not be reachable from the source vertex s, if such a cycle exists in the graph G. We give the Algorithm B here.

Algorithm B:

Input: a weighted, directed graph $G = (V, E)$ with edge weight function W, and a vertex s as the source vertex.

Output: a negative-weight cycle in the graph G that may not be reachable from the source vertex s, if such a cycle exists in the graph G.

Step 1. From the graph $G = (V, E)$, constructs a new graph $G' = (V', E')$ as follows.

Step 1.1. Let $V' = V \cup s', t'$, where s' and t' are two new vertices added to the graph $G = (V, E)$.

Step 1.2. For each $v \in V$, add one new directed edge (s', v) to the graph $G = (V, E)$ and let the edge weight $W(s', v) = 1$.

Step 1.3. For each $v \in V$, add one new directed edge (v, t') to the graph $G = (V, E)$ and let the edge weight $W(v, t') = 1$.

Step 2. Call the Algorithm A on the new graph $G' = (V', E')$ with s' as the source vertex. Return any negative cycle if found.

Theorem 4.1 *The above Algorithm B is correct. That is, Algorithm B could find a negative-weight cycle in the given weighted, directed graph G that may not be reachable from the source vertex s, if such a cycle exists in the graph G.*

PROOF. A sketch of the proof is provided here. Since the newly added vertex s' has a directed edge to each vertex of the graph $G = (V, E)$, the Algorithm B can find one negative-weight cycle that may not be reachable from the source vertex s in the original graph G, if such a cycle exists in the graph G. □

Now we analysis the time complexity of Algorithm B. For the new graph $G' = (V', E')$, we can see $|V'| = |V| + 2 = O(|V|)$, and $|E'| = |E| + 2|V|$. The time complexity of the Algorithm B is the time for the call of Algorithm A on the new graph $G' = (V', E')$, which is bounded by $O(|V'||E'|) = O(|V||E| + |V||V|)$.

The time complexity of the Algorithm B is still $O(|V||E|)$, suppose the graph $G = (V, E)$ is a connected graph.

Therefore, we have

Theorem 4.2 *Given a weighted, directed graph $G = (V, E)$, the Algorithm B, in time $O(|V||E|)$, could find a negative-weight cycle in the given weighted, directed graph G that may not be reachable from the source vertex s, if such a cycle exists in the graph G.*

5 Summary

This report presents the algorithms for finding a negative-weight cycle in a weighted directed graph. The approaches are based on the Bellman-Ford algorithm for the single-source shortest-path problem. The time complexity of the approaches is not worse than that of the Bellman-Ford algorithm.

References

[1] Chen, J., *Analysis of Algorithms Class Lecture*, Texas A&M University, 2003.

[2] Cherkassky B. V., Goldberg, A. V., Negative-Cycle Detection Algorithms, *Mathematical Programming*, Springer-Verlag, vol. 85, pp. 277311, 1999.

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., *Introduction to Algorithms*, Second edition, MIT Press/McGraw-Hill, 2001.

[4] Design of algorithm class, www.cs.bgu.ac.il/ algo012/HW/4.ps, 2001.

[5] www.andrew.cmu.edu/user/adityaa/211 /LectureGraphs_III.ppt