RAIRO-Theor. Inf. Appl. 43 (2009) 567–583
DOI: 10.1051/ita/2009011

Available online at:
www.rairo-ita.org

LABELED SHORTEST PATHS IN DIGRAPHS WITH NEGATIVE AND POSITIVE EDGE WEIGHTS [*]

Phillip G. Bradford[1,**] and David A. Thomas[2]

Abstract. This paper gives a shortest path algorithm for CFG (context free grammar) labeled and weighted digraphs where edge weights may be positive or negative, but negative-weight cycles are not allowed in the underlying unlabeled graph. These results build directly on an algorithm of Barrett *et al.* [*SIAM J. Comput.* **30** (2000) 809–837]. In addition to many other results, they gave a shortest path algorithm for CFG labeled and weighted digraphs where all edges are nonnegative. Our algorithm is based closely on Barrett *et al.*'s algorithm as well as Johnson's algorithm for shortest paths in digraphs whose edges may have positive or negative weights.

Mathematics Subject Classification. 68Q25, 52B05, 68Q42, 05C78.

1. Introduction and motivation

Barrett *et al.* [1–3] use labeled graphs to model travel through multiple modes of transportation. Different modes of transportation are associated with different labels. Using the expressive power of formal languages, only specific combinations of multimodal transportation may be allowed. In some cases, different modes of transportation give different net gains or losses as compared to other options.

Keywords and phrases. Shortest paths, negative and positive edge weights, context free grammars.

[*] *Partially supported by NSF grant 0244156 "Collaborative Research: REU Site for Pervasive and Mobile Computing".*

[1] Department of Computer Science, The University of Alabama, Box 870290, Tuscaloosa, AL 35487-0290, USA; `pgb@cs.ua.edu`

[2] Mercer University, Department of Computer Science, 1400 Coleman Ave, Macon, GA 31207, USA; `David.A.Thomas@student.Mercer.edu`

[**] *Partially supported by a University of Alabama Research Advisory Committee (RAC) grant.*

Graphs with negative edge weights allow these models to compare and contrast gains and losses by intermixing different transportation options.

For example, suppose your employer is reimbursed for your travel from point x to point y at a fixed rate per mile traveled directly reflecting your car's travel costs. Reimbursement is only for the shortest path from x to y where all edges are weighted at your car's reimbursement rate. Shortest paths from x to y can be found by computing the shortest paths on a transportation graph while weighting each edge e with the product of its length and the reimbursement rate. Thus if you have opportunity to use a variety of different modes of travel (public transit, cab, train, etc.) you may re-weight the edges to reflect the net amount gained or lost while intermixing various transport options. For example, a link of your travel using your car has net cost 0, a link traveled by limousine will have a positive net cost, a link traveled by public transit may run at $1/3$ your car's cost per mile giving it net cost of $-2/3$ per mile, etc. So a shortest path from x to y in such a net-cost augmented graph may include both positive and negative labeled edge weights. Finally, in such transportation graphs, specific combinations of multimodal transit may be allowed or restricted using path labels.

Intuitively, this paper applies the Bellman-Ford algorithm to detect certain negative-length paths in labeled graphs in a Johnson-algorithm framework [6,9]. Also, by applying an efficient algorithm of Barrett *et al.* along with Johnson's algorithm this paper gives an efficient way to find shortest paths in graphs with negative edge weights.

1.1. Background

A *labeled directed graph* (LDG) $G = (\Sigma, V, E)$ is a multigraph consisting of a set V of vertices and the edge set

$$E \subseteq V \times V \times (\Sigma \cup \{\epsilon\}) \times \mathbb{R}$$

of labeled, directed, and weighted edges. All edge weights are represented or approximated in a way that each arithmetic and comparison operation takes constant time.

Given an edge $e = (u, v, t, r) \in V \times V \times (\Sigma \cup \{\epsilon\}) \times \mathbb{R}$, then e is from node u to node v, this edge is labeled by the symbol t and its cost is r. An edge (u, v, ϵ, r) is unlabeled.

An LDG G is a multigraph that may have several edges (all labeled differently) between any two distinct nodes. No self-loops are allowed. That is, take two edges $e_1 = (u, v, t_1, r_1)$ and $e_2 = (u, v, t_2, r_2)$. If $t_1 = t_2$, then $u \neq v$ and $r_1 = r_2$. This also holds for $t_1 = t_2 = \epsilon$ (the unlabel).

Definition 1.1. Given an LDG $G = (\Sigma, V, E)$, for any pair of nodes $(u, v) \in V \times V$, take the set

$$T(u, v) \quad = \quad \{(u, v, t_i, r_i) : (u, v, t_i, r_i) \in E\}.$$

The LDG G's underlying unlabeled subgraph is $G_1 = (V, E_1)$, where $(u, v, r_i) \in E_1$ iff

$$r_i = \min\{r_j : (u, v, t_j, r_j) \in T(u, v)\}.$$

Given any LDG G, this paper assumes its underlying unlabeled subgraph $G_1 = (V, E_1)$ has no negative cycles. This may be checked, after some minor preprocessing, using the Bellman-Ford algorithm in $O(|V||E|)$ operations [6].

Given an edge $e = (u, v, t, r)$, then the edge label and edge weight functions are

$$l(e) = t \qquad \text{and} \qquad w(e) = r.$$

If $e = (u, v, t, r)$, then by the uniqueness of edge labels $w(u, v, t) = r$.

A path p is completely described by its edges $e_s \to \cdots \to e_t$ all in E. The path p has the label

$$l(p) = l(e_s) \ \ldots \ l(e_t).$$

The path p has cost

$$w(p) = \sum_{i=1}^{t} w(e_i).$$

A context-free grammar $\mathcal{G} = (N, \Sigma, P, S)$ consists of a set of nonterminals N, a set of terminals Σ, a set of productions P and the start symbol S which is a nonterminal. Let $\epsilon \notin \Sigma$, be the empty symbol.

An LDG G has a grammar $\mathcal{G}$ associated with it. This paper assumes all grammars are given in Chomsky normal form (CNF).

Given a context-free grammar $\mathcal{G}$ and a path p and its edge label $l(p)$, then we say p is a *validly labeled path* iff $l(p) \in L(\mathcal{G})$, see for example [8].

Definition 1.2. Given an LDG $G = (\Sigma, V, E)$ and let $u, v \in V$, then

$$p = u \overset{A}{\rightsquigarrow} v$$

means there is a valid labeled path p starting from nonterminal A using $L(G)$ from node u to node v in G.

Since this paper assumes all CFGs are in Chomsky normal form (CNF), all of the productions P of any CFG $\mathcal{G} = (N, \Sigma, P, S)$ are of the form:

$$\begin{aligned} S &\longrightarrow \epsilon \\ A &\longrightarrow BC \\ D &\longrightarrow x \quad \text{for some } x \in \Sigma \end{aligned}$$

where S is the start symbol. The elements S, A, B, C and D are non-terminals and the production $(S \longrightarrow \epsilon) \in P$ iff the grammar $\mathcal{G}$ can generate the empty string ϵ. Moreover, if $\mathcal{G}$ can generate the empty string, then the start-symbol S cannot be on the right-side of any production.

The issue of unlabeled paths can be treated separately from labeled paths. Consider an LDG $G = (\Sigma, V, E)$ and its associated grammar $\mathcal{G}$ where this grammar $\mathcal{G}$ can generate the empty symbol.

To find unlabeled paths separately from finding labeled paths follow the next steps:

1. create the subgraph $G' = (V, E')$ of G, where both G' and G have the same vertices;
2. the graph G' has edges that are exactly the unlabeled edges of G. That is, $E' = \{e : l(e) = \epsilon \wedge e \in E\}$;
3. run a standard all-pairs shortest path algorithm on G' that allows edge weights to be negative or positive, while ignoring the ϵ labels. For example, Johnson's algorithm [6,9] or Floyd-Warshall [6] will do.

Thus, since the unlabeled case can be done as just illustrated, from here, on all edges are labeled with terminal symbols.

1.2. Previous work

Mendelzon and Wood [10] give database applications of labeled shortest path problems. They give complexity results for regular expression labeled simple paths.

In addition to numerous other significant results, Barrett *et al.*'s important paper [1] gave two algorithms for finding shortest paths in LDGs with nonnegative edges weights. Their first algorithm, called BJM from here on, costs $O(|V|^5|N|^2|R|^2)$ operations and the second algorithm, Fast-BJM from here on, costs $O(|V|^3|N||R|)$. Their proof-of-correctness holds only for nonnegative edges. It seems they deliberately chose to exclude negative edge weights since negative edge weights were not directly applicable to their research. Nonetheless, our first result shows Barrett *et al.*'s $O(|V|^5|N|^2|R|^2)$ algorithm works on LDGs with negative or positive edge weights but no negative-weight cycles. Furthermore, we point out that Barrett *et al.*'s $O(|V|^3|N||R|)$ algorithm does not work for LDGs with negative edge weights. Barrett *et al.* never suggested their algorithms worked for negative edge weights. To work with negative and positive edge weights we give a rendition of Johnson's algorithm using Fast-BJM. Our new algorithm is Johnson-Fast-BJM and it assumes there are no negative cycles in the underlying subgraph, see Definition 1.1.

Yannakakis [13], page 237, points out that Valiant's Boolean matrix multiplication context-free word recognition algorithm can be used for single-source labeled path reachability in DAGs. Bradford and Choppella [4] give an algorithm, using the exact-path problem [11], to find shortest paths in LDGs where the edge labels are from Dyck and semi-Dyck languages. Their algorithm [4] costs $O(|V|^\omega \log n)$ where ω is the exponent for multiplying two $|V| \times |V|$ Boolean matrices. For example, to date [5,6], the smallest value for ω is about 2.376.

Greenlaw *et al.* [7] discuss several formal-language based reachability problems that are $\mathcal{P}$-complete. For example, the LGAP problem is a semi-Dyck language constrained reachability problem on a directed graph G that is $\mathcal{P}$-complete. Ruzzo uses a simulation by a one-way non-deterministic push-down automata to show the acyclic LGAP problem is LOGCFL [7,12].

2. Shortest paths in labeled directed graphs

There are many practical shortest path algorithms [6,14]. The basic $(\min, +)$ algorithm solves the all-pairs shortest path distance problem. There are more efficient algorithms for the single-source problem with non-negative edge weights (Dijkstra's algorithm) or sparse graphs (Johnson's algorithm). See Zwick [14] for a recent survey on shortest path algorithms, including approximation algorithms. To the best of our knowledge, other than work mentioned in Section 1.2, there has not been a lot of work on shortest path problems for labeled and weighted graphs.

Given an LDG G, its underlying unlabeled subgraph $G_1 = (V, E_1)$, and its associated context-free grammar $\mathcal{G}$, this paper assumes G_1 has no negative cycles. Since G_1 has no negative cycles, then G cannot have any labeled negative cycles.

Lemma 2.1. *Given an LDG $G = (V, E)$ with its underlying unlabeled subgraph $G_1 = (V, E_1)$, and its associated context-free grammar $\mathcal{G}$. If G_1 has no negative cycles, then G cannot have any labeled negative cycles.*

Proof. For the sake of a contradiction, suppose G_1 has no negative cycles, but G has at least one labeled negative cycle. Let this cycle be

$$p \quad = \quad v_0 \overset{A}{\rightsquigarrow} v_0$$

and assume p is made of more than one node.

Given G consider constructing G_1. Each time two nodes u and v are connected by edges in E, then there is an edge $(u, v, r) \in E_1$ where (u, v, r) is of minimal weight among all labeled edges connecting u and v in E by Definition 1.1. This is regardless of (u, v, r)'s original label in G. Therefore, the unlabeled edges in G_1 representing p must form a negative cycle in G_1, giving a contradiction and completing the proof. □

Barrett *et al.*'s algorithms are based on the next definition and the recurrence following it.

Definition 2.1 (Barrett *et al.* [1])**.** Given an LDG $G = (\Sigma, V, E)$ and let A be a nonterminal belonging to context free language associated with G.

Define $D(u, v, A)$ as the approximate shortest path distance from node u to node v where this path is constructed according to the language derived from the nonterminal A.

Intuitively, if $p = u \overset{A}{\rightsquigarrow} v$ and the cost of p is finite, then $D(u, v, A)$ starts out as an upper-bound on the cost of p and converges down to the actual cost of p as the labeled path algorithms run.

Given a LDG $G = (\Sigma, V, E)$, Barrett *et al.* [1] give the next recurrence for computing $D(u, v, A)$:

$$D(u, v, A) \quad = \quad \min_{A \to BC} \left\{ \min_{k \in V} \{ D(u, k, B) + D(k, v, C) \} \right\}$$

and for a fixed literal t,

$$D(u, v, t) \quad = \quad \begin{cases} w(u, v, t) & \text{if } (u, v, t) \in E \\ +\infty & \text{otherwise.} \end{cases}$$

Barrett *et al.* [1], Observation 15, give several algorithms for LDGs. Their algorithms culminate in Fast-BJM costing $O(|V|^3|N||R|)$ for solving the all-pairs shortest distances problem.

Algorithm 1 Initialize the APSD LDG algorithms

```
// Given an LDG $G = (\Sigma, V, E)$ where the grammar is in CNF.
1. Initialize-Matrix_D($G$)
2. for all pairs $(u, v) \in V \times V$ do
3.         for all nonterminals $A \in N$ do
4.                 $D(u, v, A) \leftarrow +\infty$
5. for all edges $(u, v) \in E$ do
6.     for all productions $A \to t$ where $t \in \Sigma$ do
7.         if $l(u, v) = t$ then
8.                 $D(u, v, A) \leftarrow D(u, v, t) \leftarrow w(u, v, t)$
9. Return $D$
```

Theorem 2.1 gives our analysis of the BJM algorithm, which is Algorithm 2. Theorem 2.1 indicates that the BJM algorithm works on LDGs with edges having either negative or positive edge weights. Barrett *et al.* [1], page 817, describe their algorithm as a Bellman-Ford-type algorithm. However, it seems in the context of their Observation 15 and its surrounding discussion that their use of a Bellman-Ford-type algorithm was focused on how the productions of the grammar are split and not the values of the digraph's edges.

Definition 2.2. Given an LDG $G = (\Sigma, V, E)$ and let $u, v \in V$ and say $u \overset{A}{\rightsquigarrow} v$. Let $\delta(u, v, A)$ be the shortest labeled-path distance from u to v whose label can be derived from nonterminal A.

Let $G = (\Sigma, V, E)$ be an LDG. Given Definition 2.2, if $u, v \in V$, then $u \overset{A}{\not\rightsquigarrow} v$ indicates there is no valid labeled-path from u to v in G. That is, $u \overset{A}{\not\rightsquigarrow} v$ means $D(u, v, A) = \delta(u, v, A) = +\infty$.

Barrett *et al.* [1] indicate the following result holds, but in their context LDGs have nonnegative edges.

Algorithm 2 BJM: Barrett *et al.*'s $O(|V|^5|N|^2|R|)$ algorithm [1]

```
// Given an LDG $G = (\Sigma, V, E)$ where the grammar is in CNF.
1. BJM($G$)
2. Initialize-Matrix_D($G$)
3. **Repeat** $|V|^2|N|$ **Times**
4.   **for all** pairs $(u, v) \in V \times V$ **do**
5.       **if** $u \neq v$ **then**
6.           **for all** nonterminals $A \in N$ **do**
7.               $D(u, v, A) = \min_{A \to BC} \{ \min_{k \in V} \{D(u, k, B) + D(k, v, C)\ \}\}$
8. Return $D$
```

Lemma 2.2. *Given an LDG $G = (\Sigma, V, E)$ with positive and negative edge weights and no negative cycles. Consider the set*

$$\mathcal{D} \quad = \quad \{D(u, v, A) : u \overset{A}{\rightsquigarrow} v \ \wedge \ D(u, v, A) > \delta(u, v, A)\}$$

then executing lines 4–7 of BJM (Algorithm 2) sets the value of at least one $D(u, v, A) \in \mathcal{D}$ to $\delta(u, v, A)$ thus removing $D(u, v, A)$ form $\mathcal{D}$.

Proof. This paper assumes the grammar of $L(G)$ is in Chomsky normal form.

All $D(u, v, t)$ where t is a terminal cannot change, thus $D(u, v, t) = \delta(u, v, t)$. Therefore, no such $D(u, v, t)$ can be in $\mathcal{D}$. Similarly, if the nonterminal, say, B is in a production of the form $B \to b$ for some terminal b, then $D(u, v, B)$ is fixed and cannot be in $\mathcal{D}$. Finally, for any u, v and nonterminal A if there is *no* labeled path from u to v that starts from A, then $D(u, v, A) = +\infty$ and this is also minimal. Thus, if $u \overset{A}{\not\rightsquigarrow} v$ for nonterminal A, then $D(u, v, A) \notin \mathcal{D}$.

The only remaining productions from the Chomsky normal form CFG are of the form $A \to BC$ where B and C are nonterminals. This leads to the next claim:

Claim 2.1. If $\mathcal{D} \neq \emptyset$ then there must be at least one $D(u, v, A) \in \mathcal{D}$ so that there is a production $A \to BC$ and some $k \in V$ where

$$D(u, k, B) \notin \mathcal{D} \quad \text{and} \quad D(k, v, C) \notin \mathcal{D}.$$

Suppose for the sake of a contradiction that this claim does not hold. In particular, say for every $D(u, v, A) \in \mathcal{D}$ so that all productions $A \to BC$ and all $k \in V$ are so that

$$D(u, k, B) \in \mathcal{D} \quad \text{or} \quad D(k, v, C) \in \mathcal{D}.$$

Since $D(u, v, A) \in \mathcal{D}$, then $u \overset{A}{\rightsquigarrow} v$ holds and let p be this validly-labeled path from u to v in G. The CNF grammar starting from the nonterminal A can be represented as a binary tree T all of whose internal nodes are nonterminals and all external nodes (leaves) are terminals. These external nodes label each of the edges in the path p and p is a simple path because G has no negative cycles. Let X be the deepest nonterminal in N derivable from A such that $D(u_i, v_j, X) \in \mathcal{D}$ for some u_i, v_j in the path p. Without loss, say $X \to YZ$, then both nonterminals Y and Z must be so that $D(u_i, k, Y) \notin \mathcal{D}$ and $D(k, v_i, Z) \notin \mathcal{D}$ for some $k \in V$ giving a contradiction thus substantiating the claim.

Now, say $\mathcal{D} \neq \emptyset$, then by Claim 2.1 there is some $D(u, v, A) \in \mathcal{D}$ so that there is a production $A \to BC$ and some $k \in V$ where $D(u, k, B) \notin \mathcal{D}$ and $D(k, v, C) \notin \mathcal{D}$, thus the minimization in line 7 of Algorithm 2 sets the value of $D(u, v, A)$ to $D(u, k, B) + D(k, v, C)$ which must be $\delta(u, v, A)$, completing the proof. □

Theorem 2.1. *BJM (Algorithm 2) works on LDGs whose weighted edges may be either positive or negative, but without negative-weight cycles.*

Proof. Given an LDG $G = (\Sigma, V, E)$ where the edge weights may be negative or positive but there are no negative cycles.

By Lemma 2.2 at least one array element $D(u, v, A)$ is set as

$$D(u, v, A) \quad \leftarrow \quad \delta(u, v, A)$$

in each run of lines 4-7 of BJM (Algorithm 2).

Since BJM loops $|V|^2|N|$ times at line 3, then for all $u, v \in V$ and all $A \in N$ it must be that $D(u, v, A) = \delta(u, v, A)$.

This completes the proof. □

Once BJM is run on an LDG G, then negative cycles can be discovered by running CheckForNegativeCycles(G, D) as given in Algorithm 3. Clearly, this algorithm costs $O(|V|^3|N||R|)$.

Algorithm 3 Checking for negative cycles

```
// Given an LDG $G = (\Sigma, V, E)$ where the grammar is in CNF and
//  say BJM (Algorithm 2) has *already computed* the $D(u, v, A)$ values
1. CheckForNegativeCycles$(G, D)$
2.   **for all** pairs $(u, v) \in V \times V$ **do**
3.       **for all** nonterminals $A \in N$ **do**
4.           $D(u, v, A) = \min_{A \to BC} \{\min_{k \in V}\{D(u, k, B) + D(k, v, C)\}\}$
5. Return $D$
```

Theorem 2.2. *Given a LDG $G = (\Sigma, V, E)$ with negative or positive edge weights and suppose BJM(G) returns the set $\{D(u, v, A) : u, v \in V \wedge A \in N\}$. If any $D(u, v, A)$ decreases during a run of CheckForNegativeCycles(G, D), then there is a negative cycle in G.*

Proof. Suppose there is a validly labeled negative cycle in the LDG G:

$$p \quad = \quad u \overset{A}{\rightsquigarrow} u.$$

Since we do not allow self-loops for any node, it must be that there is a production $A \rightarrow BC$ where the path p is comprised of two shortest labeled paths

$$\begin{aligned} p_1 &= u \overset{B}{\rightsquigarrow} v \\ p_2 &= v \overset{C}{\rightsquigarrow} u \end{aligned}$$

so that $w(p_1)+w(p_2) < 0$. Both $D(u,v,B)$ and $D(v,u,C)$ must have been returned by BJM by Theorem 2.1.

However, this means for some production $A \rightarrow BC$, it must be that

$$D(u,u,A) \quad > \quad \min_{A \rightarrow BC}\{\min_{k \in V}\{D(u,k,B)+D(k,u,C)\}\},$$

which will be detected by CheckForNegativeCycles(G,D) in Algorithm 3. □

3. Barrett et al.'s $O(|V|^3|N||R|)$ algorithm

This section briefly reviews the Fast-BJM algorithm (Algorithm 4). This algorithm assumes a min heap-structure keyed off of the $D(u,v,A)$ values. This is very reminiscent of the min heap used by Dijkstra's algorithm. The function **Initialize-Heap_H(G)** on line 3 simply puts the D values in an array of length $|V|^2|N|$.

The next result is due to Barrett *et al.* [1]. We refer the reader to [1] for the proof.

Theorem 3.1 (See [1])**.** *Given an LDG $G = (\Sigma, V, E)$ where all edge weights are nonnegative. At the termination of the algorithm Fast-BJM, $D(u,v,A) = \delta(u,v,A)$ for all $(u,v,A) \in V \times V \times N$.*

This may be proved using the next loop invariant:

I1: At the start of each iteration of the **while** loop on line 5 of Figure 4, $D(u,v,A) = \delta(u,v,A)$ for all $D(u,v,A) \in S$.

3.1. Fast-BJM fails on LDGs with negative edge weights

Fast-BJM does not work on LDGs with negative edge weights as is shown here with a concise example. This example is supplied since in Theorem 2.1 we showed BJM (Algorithm 2) does correctly find shortest labeled paths on LDGs whose weighted edges may be positive or negative. Again, we mention, Fast-BJM was not intended to work on LDGs with negative edge weights, see [1].

Algorithm 4 Fast-BJM: Barrett *et al.*'s $O(|V|^3|N||R|)$ algorithm [1]

```
1. Fast-BJM($G$) // where $G = (\Sigma, V, E)$
2. Initialize-Matrix_D($G$)
3. Initialize-Heap_H($G$)
4. $S_1 \leftarrow \emptyset$
5. while $H \neq \emptyset$ do
6.   $D(u, v, X) \leftarrow$ extractMin($H$)
7.   $S_1 \leftarrow S_1 \cup \{D(u, v, X)\}$
8.   for all productions of the form $A \rightarrow BC$ do
9.       if $B = X$ then
10.          for all vertices $v_0 \in V$ do
11.              $L \leftarrow P(u, v_0, A)$
12.              $val \leftarrow D(u, v, B) + D(v, v_0, C)$
13.              if $val < D(L)$ then
14.                  decreaseKey($H, L, val$)
15.      if $C = X$ then
16.          for all vertices $u_0 \in V$ do
17.              $L \leftarrow P(u_0, v, A)$
18.              $val \leftarrow D(u_0, u, B) + D(u, v, C)$
19.              if $val < D(L)$ then
20.                  decreaseKey($H, L, val$)
21. Return $D$
```

Consider the following grammar (S is the start symbol):

$$\begin{array}{lcl} S & \rightarrow & \epsilon \\ S & \rightarrow & a \\ S & \rightarrow & AA \\ A & \rightarrow & AA \\ A & \rightarrow & a \end{array}$$

which gives the language $\{a^n : n \geq 0\}$.

The LDG in Figure 1 is a very basic case where Fast-BJM fails to find correct shortest path distances.

The objective is for Fast-BJM to compute shortest path distances from u to v derived from A. Fast-BJM first removes the element $D(u, v, A) = -5$ from its heap H and adds it to the set S_1. However, this violates invariant **I1** in the proof of Theorem 3.1. For lines 12 or 18 in Algorithm 4 cannot reduce the value of $D(u, v, A)$. Thus, the algorithm will terminate with $D(u, v, A) = -5$, while the actual shortest path from u to v derived from A has weight -6.

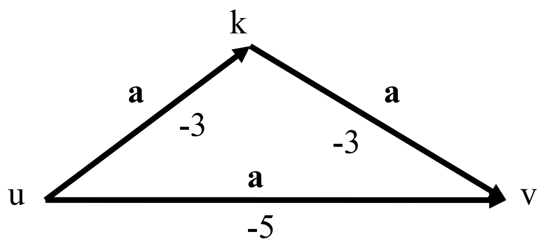

FIGURE 1. A labeled graph with negative edge weights.

3.2. AUGMENTING FAST-BJM TO WORK WITH LDGS WHERE EACH EDGE MAY HAVE POSITIVE OR NEGATIVE WEIGHT

It is well known that Johnson's algorithm [9] finds shortest paths for graphs with negative edge weights but having no negative cycles [6,9]. Johnson's algorithm also solves the all-pairs shortest path problem efficiently for sparse graphs.

Here we give an algorithm, Johnson-Fast-BJM, that works on LDGs where each edge may have negative or positive weight, but negative weight cycles are not allowed in the underlying unlabeled subgraphs. More precisely, our algorithm works for LDGs whose underlying unlabeled subgraphs do not have negative cycles, see Definition 1.1.

A key challenge is justifying and adjusting Johnson's reweighting scheme for several differently labeled edges between any pair of nodes. We build a graph consisting of the lowest-cost edges between each pair of vertices while ignoring the edge-label constraints. The Bellman-Ford algorithm is applied to this cheapest edge-cost graph to find the shortest possible path from the source vertex to all other vertices. Then Johnson's reweighting scheme can be applied using these shortest-paths. This works due to a minor variation of the classical triangle inequality which holds since we do not allow the unlabeled graph to have negative cycles. Intuitively, edge label constraints can only lead to more costly paths than those in such lowest-cost edge graphs.

Johnson-Fast-BJM is essentially an application of Barrett *et al.*'s Fast-BJM in a Johnson-style framework. Johnson-Fast-BJM also uses the Bellman-Ford algorithm on the underlying unlabeled subgraph to detect negative cycles in the underlying graph. Johnson-Fast-BJM's cost is dominated by Fast-BJM which is $O(|V|^3|N||R|)$ time. Next we discuss both the intuition and mechanics of the algorithm Johnson-Fast-BJM. A proof of correctness is given in Theorem 3.2.

Consider a graph G with CFG $\mathcal{G}$ and first apply Johnson's reweighting technique [6,9]. That is, construct a new weighted and labeled digraph G':

1. Create G' vertex set $V(G') \leftarrow V(G) \cup \{s\}$ where $s \notin V(G)$ is a new node.
2. Create the edge set $E(G') \leftarrow E(G) \ \cup \ \{(s, v, x, 0) \ : v \in V(G') \wedge x \notin \Sigma\}$. That is, $(s, v, x, 0)$ is a zero-weight x-labeled directed edge from s to v where x is not a previously used edge-label. Augment the CFG $\mathcal{G}$ so that

each string in $L(\mathcal{G})$ starts with a single occurrence of the new symbol x. All other edges in $E(G')$ inherit their labels and weights from $G(E)$.

Any pair of nodes (u, v) in the new graph G' have weight,

$$w'(u,v) \quad = \quad \begin{cases} \min_{t\in\Sigma}\{w(u,v,t)\} & \text{if there is at least one edge from } u \text{ to } v, \\ +\infty & \text{otherwise.} \end{cases}$$

Consider an LDG $G = (\Sigma, V, E)$ whose edge weights may be either positive or negative. Check G's underlying unlabeled subgraph, in lines 4 to 8 of Algorithm 5, for any negative cycles. If there is a negative cycle in G's underlying unlabeled subgraph, then report it and stop. Otherwise, we continue following the structure of Johnson's algorithm.

In the graph G', let $\delta_{BF}(s, v)$ be the shortest path distance between s and v found by applying the Bellman-Ford algorithm to the underlying unlabeled subgraph using the weight function w'.

The triangle inequality holds for the weight function w' as computed by Bellman-Ford in Johnson-Fast-BJM:

$$\delta_{BF}(s,v) \quad \leq \quad \delta_{BF}(s,u) + \min_{t\in\Sigma}\{w(u,v,t)\} \quad \text{(triangle inequality)}$$

for all edges $(u, v) \in E[G]$. Proving this can be done by contradiction. For the sake of a contradiction, suppose

$$\delta_{BF}(s,v) \quad > \quad \delta_{BF}(s,u) + \min_{t\in\Sigma}\{w(u,v,t)\} \tag{1}$$

but this contradicts the fact that $\delta_{BF}(s, v)$ is a shortest path value between s and v using edges with minimal weights (including edges with negative weights). No matter what other weight is considered for an edge between u and v the triangle inequality still holds. This includes $w(u, v, t) = +\infty$ for all $t \in \Sigma$, for example when there is no valid CFG labeled path from u to v. Note, that Bellman-Ford will return FALSE iff Equation (1) holds for any $v \in V$.

Take the assignment $h(v) \leftarrow \delta_{BF}(s, v)$ as given in line 10 of Johnson-Fast-BJM. Next, define a new weight function [1,9]:

$$\widehat{w}(u,v,t) \quad = \quad w(u,v,t) + h(u) - h(v).$$

Moreover, it must be that $\widehat{w}(u, v, t) \geq 0$ for all edges $(u, v, t) \in E[G]$, otherwise we will violate the triangle inequality above. This will be formally shown in the next proof.

Then run the Fast-BJM algorithm on the LDG G but with the augmented weight function $\widehat{w}$. At the termination of Fast-BJM, then $\widehat{\delta}(u, v, A)$ will be equal to $D(u, v, A) - h(u) + h(v)$. Lines 14 and 15 re-adjust these values back to $D(u, v, A)$ for all nonterminals $A \in N$, completing the algorithm.

The proof of the next theorem closely follows the proof of correctness of Johnson's algorithm as given in Cormen *et al.* [6].

Algorithm 5 Johnson-Fast-BJM algorithm

```
// Given an LDG $G = (\Sigma, V, E)$ with positive or negative edge weights
// and associated CFG $\mathcal{G} = (N, \Sigma, P, S)$
// Only works if $G$'s underlying unlabeled subgraph has no negative
weighted cycles
1. Johnson-Fast-BJM($G$)
2. Take a node $s \notin V$ and a new nonterminal $x \notin \Sigma$ and compute $G'$:
3.     $V[G'] \leftarrow V[G] \cup \{s\}$ and $E[G'] \leftarrow E[G] \cup \{(s, v, x, 0) : v \in V[G]\}$
4. Add nonterminals $S', X \notin N$ to $N$ and the rules $S' \rightarrow XS$ and
$X \rightarrow x$ to $P$
5. Make $S'$ the new start symbol of $\mathcal{G}$
6. for all $(u, v, t) \in E[G]$ do
7.     $w'(u, v) \leftarrow \min_{t \in \Sigma}\{w(u, v, t)\}$
// Replacing the weight function $w$ with $w'$ from Bellman-Ford
8. if Bellman-Ford($G'(w'), s$) = FALSE then
9.     print "$G$'s underlying unlabeled subgraph contains a negative
weight cycle"
10.    Exit;
// Bellman-Ford gives $\delta_{BF}(s, v)$ for all $v \in V[G']$ ignoring edge labels in $G'$
11. for each vertex $v \in V[G']$ do
12.    $h(v) \leftarrow \delta_{BF}(s, v)$
13. for each edge $(u, v, t) \in E[G']$ do
14.    $\widehat{w}(u, v, t) \leftarrow w(u, v, t) + h(u) - h(v)$
15. $\widehat{\delta} \leftarrow$ Fast-BJM($G(\widehat{w})$) // LDG $G$ with the weight function $\widehat{w}$ replacing $w$
16. for all $(u, v, A) \in V[G] \times V[G] \times N$ do
17.    $D(u, v, A) \leftarrow \widehat{\delta}(u, v, A) + h(v) - h(u)$
18. Return $D$
```

Theorem 3.2 (Correctness of Johnson-Fast-BJM)**.** *Given an LDG G whose edge weights may be positive or negative, but G's underlying unlabeled subgraph has no negative cycles. Johnson-Fast-BJM (Algorithm 5), computes the shortest labeled path values in G in $O(|V|^3|N||R|)$ operations. Moreover, it detects negative weight cycles in the underlying unlabeled graph and terminates.*

Proof. First, lines 4 through 8 of Johnson-Fast-BJM will detect and report any negative cycles in G's underlying unlabeled subgraph. If there are any negative cycles in G's underlying unlabeled subgraph, then Johnson-Fast-BJM reports this and terminates.

The transformation between w and $\widehat{w}$ is classical. A proof of correctness can be found in [6,9]. We recall these relevant facts here for completeness.

In line 6 of Algorithm 5 the Bellman-Ford algorithm computes (ignoring edges' labels) all shortest path values from s. Then, as in lines 11 and 12, for all $(u, v, t) \in V(G) \times V(G) \times \Sigma$, let

$$\widehat{w}(u, v, t) \quad = \quad w(u, v, t) + h(u) - h(v).$$

That is, $h(x) = \delta_{BF}(s, x)$ for all $x \in V[G]$, which is computed by Bellman-Ford ignoring edge labels and using the weight function $w'(u, v) = \min_{t\in\Sigma}\{w(u, v, t)\}$. As already noted,

$$h(u) - h(v) + \min_{t\in\Sigma}\{w(u, v, t)\} \quad \geq \quad 0$$

must hold, otherwise suppose

$$\delta_{BF}(s, u) - \delta_{BF}(s, v) + \min_{t\in\Sigma}\{w(u, v, t)\} \quad < \quad 0$$

but this violates the triangle inequality. Since by assumption, G's underlying unlabeled subgraph has no cycles, the triangle inequality holds for all $\delta_{BF}(s, v)$ computed by Bellman-Ford. Thus, $\widehat{w}(u, v, t) \geq 0$ for all $(u, v, t) \in E[G]$.

Now, we make the following claim [6,9] about the LDG G:

Claim 3.1. Consider the labeled shortest path $p = v_0 \overset{A}{\rightsquigarrow} v_k$. The path p is a shortest validly labeled path from v_0 to v_k using weight function w iff the path p is a shortest validly labeled path from v_0 to v_k using weight function $\widehat{w}$.

This claim is only different from Johnson's [9] or that found in [6] by requiring the paths to be validly labeled paths. Next, mirroring the classic proof, we prove Claim 3.1.

Thus, following the standard proof, the labeled path $p = v_0 \overset{A}{\rightsquigarrow} v_k$ has cost

$$\begin{aligned}
\widehat{w}(p) \quad &= \quad \sum_{i=1}^{k} \widehat{w}(v_{i-1}, v_i, t_i) \\
&= \quad \sum_{i=1}^{k} w(v_{i-1}, v_i, t_i) + h(v_{i-1}) - h(v_i) \\
&= \quad w(p) + h(v_0) - h(v_k),
\end{aligned}$$

by telescoping of the h values. By assumption the string $t_1 t_2 \ldots t_{k-1} \in L(\mathcal{G})$ where $\mathcal{G}$ is the context-free grammar associated with G.

Consider a validly labeled shortest path p from nonterminal A,

$$p \quad = \quad v_0 \overset{A}{\rightsquigarrow} v_k,$$

in $G(w)$ using the weight function w.

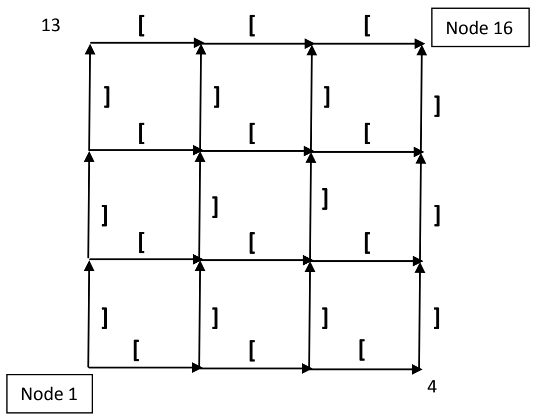

FIGURE 2. A labeled grid graph.

Now suppose there is another labeled shortest path p_1 from nonterminal A,

$$p_1 \quad = \quad v_0 \overset{A}{\leadsto} v_k$$

in $G(\widehat{w})$ using the weight function $\widehat{w}$.

Assume the paths p and p_1 have at least one different node, but both of their labels are derived from the nonterminal A.

Now by definition

$$\begin{aligned}\widehat{w}(p) &= w(p) + h(v_0) - h(v_k)\\ \widehat{w}(p_1) &= w(p_1) + h(v_0) - h(v_k)\end{aligned}$$

thus, if $\widehat{w}(p_1) < \widehat{w}(p)$, then it must be that $w(p_1) < w(p)$ meaning p is not a shortest labeled path from v_0 to v_k after all. Therefore, if p is a shortest labeled path using the weight function w, then it is a shortest labeled path using the weight function $\widehat{w}$ and vice versa. This proves Claim 3.1.

Both $\widehat{w}(u, v, t)$ is nonnegative for all $(u, v, t) \in V \times V \times \Sigma$ and $D(u, v, A) + h(u) - h(v)$ is nonnegative for all $(u, v, A) \in V \times V \times N$. Moreover, if $p = v_0 \overset{A}{\leadsto} v_k$ is a shortest labeled path from v_0 to v_k, then $w(p) = D(v_0, v_k, A)$ and $\widehat{w}(p) = D(v_0, v_k, A) + h(v_0) - h(v_k) \geq 0$ by telescoping as just shown. Therefore, Fast-BJM in line 13 of Algorithm 5 will run correctly and we can re-adjust the $\widehat{\delta}$ values in lines 14 and 15.

The algorithm Johnson-Fast-BJM(G) has running time dominated by Fast-BJM(G). Thus, the total running time of Johnson-Fast-BJM(G) is $O(|V|^3|N||R|)$. All other operations, including Bellman-Ford which costs $O(|V||E|)$, are within the $O(|V|^3|N||R|)$ bound, which completes the proof. $\square$

3.3. An example

As an example we show how the Johnson-Fast-BJM algorithm works by computing a shortest path on a directed grid graph. We follow the Algorithm 5 very closely. This grid graph has n^2 nodes for some integer n where each directed horizontal edge is labeled with an "[" while each directed vertical edge is labeled with an "]". See Figure 2. Thus, to match opening parenthesis with closing parenthesis, in going from node 1 to node 16 requires the path to be at or below the diagonal. Such a parenthesization is a semi-Dyck grammar [8].

Formally, this semi-Dyck grammar $\mathcal{G} = (N, \Sigma, P, S)$ has start-symbol S and nonterminals S, T, A, A^{-1}, the two terminals $\Sigma = \{[,]\}$ and the rules,

$$\begin{array}{rcl} S & \rightarrow & SS \\ S & \rightarrow & AA^{-1} \\ S & \rightarrow & TA^{-1} \\ T & \rightarrow & AS \\ A & \rightarrow & [\\ A^{-1} & \rightarrow &]. \end{array}$$

To start, in Figure 2 initially weight all horizontal edges with -1 and all vertical edges with -2. Now, reweight the single horizontal edge $3 \rightarrow 4$ with $+1$ and reweight the single vertical edge $4 \uparrow 8$ with -10. As expected, it is not hard to see that Fast-BJM will not work properly in this case.

However, by adding a new start node $s \notin \{1, 2, \ldots, 16\}$ and all edges $(s, i, x, 0)$, for $i \in 16 \geq i \geq 1$, where the new terminal $x \notin \Sigma$. Moreover, assuming $S' \notin N$ and $X \notin N$, then add the new nonterminals S' and X to N and make S' the start nonterminal. Finally, add the two new rules $S' \rightarrow XS$ and $X \rightarrow x$ where S was originally the start symbol. This augmentation of the grammar $\mathcal{G}$ forces all labeled paths to start from the new node s.

Next, we compute Bellman-Ford of the augmented graph while ignoring the edge labels. The graph edges weights are the lowest edge weights available discarding the labels. If the underlying graph has a negative cycle, then the algorithm halts. Otherwise we get all $h(v) = \delta_{BF}(s, v)$ values for all $v \in V$ while ignoring the labels. As was shown in the proof of correctness, the $h(v)$ values computed this way, are sufficient to allow Fast-BJM to correctly compute the shortest labeled paths (which may only cost more than the shortest un-labeled paths). Now, after normalizing the edge weights using Johnson's technique, then we run Fast-BJM as expected and finally de-normalize the shortest path weights.

References

[1] C. Barrett, R. Jacob and M. Marathe, Formal-language-constrained path problems. *SIAM J. Comput.* **30** (2000) 809–837.

[2] C. Barrett, K. Bisset, M. Holzer, G. Konjevod, M. Marathe and D. Wagner, *Label Constrained Shortest Path Algorithms: An Experimental Evaluation using Transportation Networks*. Tech. Report: Virginia Tech (USA), Arizona State University (USA), and Karlsruhe University (Germany), Presented at at the workshop on the DIMACS Shortest-Path Challenge, http://i11www.ira.uka.de/algo/people/mholzer/publications/pdf/bbhkmw-lcspa-07.pdf

[3] C. Barrett, K. Bisset, R. Jacob, G. Konejevod and M. Marathe, Classical and contemporary shortest path problems in road networks: Implementation and experimental analysis of the TRANSMIS router. *European Symposium on Algorithms (ESA 02). Lect. Notes Comput. Sci.* **2461** (2002) 126–138.

[4] P.G. Bradford and V. Choppella, Fast Dyck and semi-Dyck constrained shortest paths on DAGs (*submitted*).

[5] D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions. *J. Symb. Comput.* **9** (1990) 251–280.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *Introduction to Algorithms*, 2nd edition. MIT Press (2001).

[7] R. Greenlaw, H.J. Hoover and W.L. Ruzzo, *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press (1995).

[8] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley (1979).

[9] D.B. Johnson, Efficient algorithms for shortest paths in sparse networks. *J. ACM* **24(1)** (1977) 1–13.

[10] A.O. Mendelzon and P.T. Wood, Finding regular simple paths in graph databases. *SIAM J. Comput.* **24** (1995) 1235–1258.

[11] M. Nykänen and E. Ukkonen, The exact path length problem. *J. Algor.* **42** (2002) 41–53.

[12] W.L. Ruzzo, Complete pushdown languages. Unpublished manuscript (1979).

[13] M. Yannakakis, Graph-theoretic methods in database theory. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '90)*. ACM, New York, NY (1990) 230–242.

[14] U. Zwick, Exact and Approximate Distances in Graphs – A survey. In *Proceedings of the Ninth ESA (2001)* 33–48.

Communicated by C. Choffrut.
Received October 5, 2006. Accepted January 6, 2009.