A Note on Bertsekas' Small-Label-First Strategy

Zhi-Long Chen   Warren B. Powell*
Department of Civil Engineering & Operations Research
Princeton University
Princeton, NJ 08544, USA

April 1995
Revised: January, July 1996

*Corresponding author

Abstract

An example is presented to show that the worst-case complexity of Bertsekas' small-label-first strategy for the shortest path problem is exponential. It becomes polynomial if, when scanning a node i, its successors $j \in \Gamma(i)$ are examined in the nondecreasing order of d_{ij}, the distance between i and j.

Key words: Shortest Path Problem; Labeling Algorithm; Worst-case Complexity

Let $G = (N, A)$ be a directed graph with node set N and arc set A where the cardinality of N and A are denoted by $|N|$ and $|A|$, respectively. The nodes are numbered $0, 1, ..., |N| - 1$. Let d_{ij} denote the length of arc $(i, j) \in A$, and $\Gamma(i) = \{j|(i, j) \in A\}$ denote the successors of node i. We assume throughout this paper that each d_{ij} is nonnegative. For the well-known problem of finding a shortest path from a single origin (node 0) to each of the other nodes, most of the major algorithms first initialize a label vector $(d_0, d_1, ..., d_{|N|-1})$ and candidate list L: $d_0 = 0$; $d_i = \infty$, for each $i \neq 0$; $L = \{0\}$. Then, the algorithms repeat the following two steps until L is empty:

(a) Select a node i from the candidate list L.

(b) Remove i from L and scan node i which consists of examining each successor $j \in \Gamma(i)$ as follows: if $d_j > d_i + d_{ij}$, then update $d_j := d_i + d_{ij}$, and add node j to L if it does not belong to the current L.

Different strategies used in procedure (a) yield different algorithms. The so-called *label setting* algorithm (Dijkstra [4]) always selects a node with the smallest label from L. By contrast, the so-called *label correcting* algorithms, e.g. the Bellman-Ford algorithm [1], the D'Esopo-Pape algorithm [6], and the threshold algorithm of Glover, Klingman, Phillips and Schneider [5] use other procedures to avoid the cost of searching for the minimum label. These algorithms use a queue Q to maintain the candidate list L. At each iteration, the top node of Q is selected, removed, and scanned. They differ in the strategy for choosing the queue position to insert a node that is added to L. The worst-case complexity depends on the particular strategy used. The label setting algorithm, the Bellman-Ford algorithm, and the threshold algorithm have a worst-case complexity of $O(|N|^2)$, $O(|N||A|)$, and $O(|N||A|)$, respectively. By contrast, the D'Esopo-Pape algorithm has an exponential worst-case complexity.

Lately, Bertsekas [2] proposes a new queue insertion strategy, which he calls *Small Label First* (**SLF**) as follows:

Whenever a node j enters Q, its label d_j is compared with the label d_i of the top

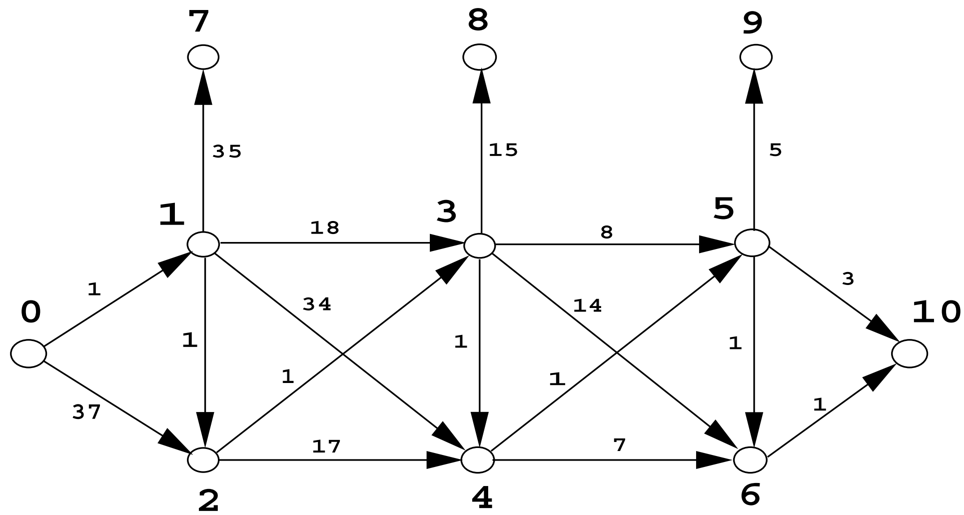

Figure 1: An example network with $|N| = 11$

node i of Q. If $d_j \leq d_i$, node j is put at the top of Q; otherwise, j is put at the bottom of Q.

Bertsekas [2] and Bertsekas, Guerriero and Musmanno [3] show through computational experiments that the **SLF** strategy is comparable with the known best shortest path algorithm, and is extremely fast if combined with the threshold method of [5]. However, the complexity of the **SLF** strategy remains as an open question [2].

In the following section, we show that the worst-case complexity of the **SLF** strategy is exponential by giving an example. In section 2, we show that if the nodes in $\Gamma(i)$ are examined in a proper order when scanning node i, then the **SLF** strategy is bounded by $O(|N|^2|A|)$. Finally, we conclude the paper in Section 3.

1 An Example Requiring Exponential Time

Figure 1 shows a network with $|N| = 11$ nodes. We apply the **SLF** strategy to this example, provided that whenever scanning a node i, the successors $j \in \Gamma(i)$ are examined in the order of nonincreasing d_{ij}. We write down each iteration of the algorithm in Table 1 where a queue is denoted by a row vector in which the leftmost component is the top

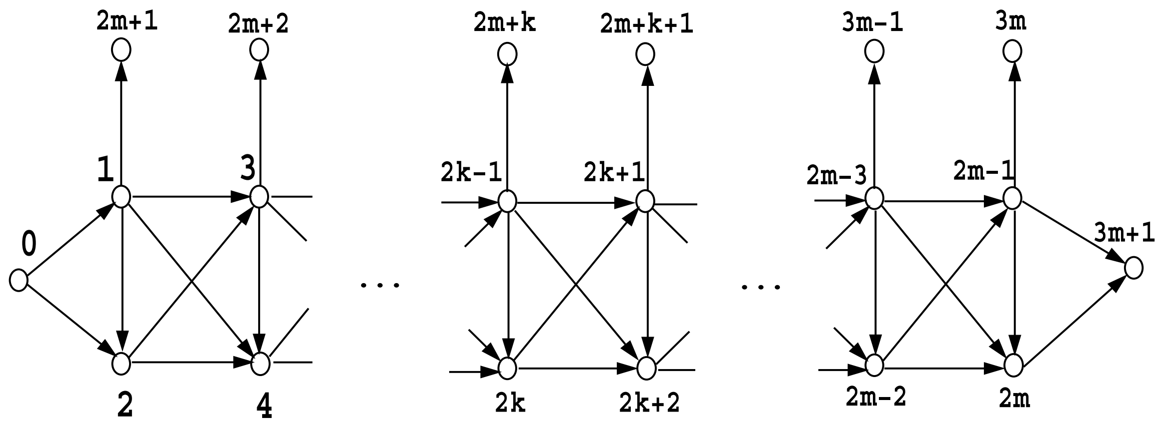

Figure 2: An example network with $|N| = 3m + 2$

node of the queue and the rightmost component is the bottom node of the queue. In this particular example, node 1 and 2 are scanned once respectively, node 3 and 4 are scanned twice respectively, and node 5 and 6 are scanned 4 times respectively.

A network with the same characteristics and of arbitrary number of nodes can be constructed as Figure 2. The network has $|N| = 3m + 2$ nodes, where m is an arbitrary positive integer. The arc lengths of the network are given in Table 2. Also suppose when scanning a node i, the successors $j \in \Gamma(i)$ are examined in the order of nonincreasing d_{ij}. Consider now the performance of the **SLF** strategy when applied to this example. We claim that upon termination of the algorithm, both node $2k - 1$ and node $2k$ ($k = 1, 2, ..., m$) are scanned exactly 2^{k-1} times. This claim is proved in the following Lemma 1 and Theorem 2.

For ease of presentation, throughout the remainder of this paper, we denote a queue by a row vector with the leftmost and rightmost components being the top and bottom nodes respectively. In the process of solving the shortest path problem for the network shown in Figure 2, the **SLF** algorithm will generate a sequence of queues and a corresponding sequence of label vectors (like those ones described in Table 1). The labels in a

iteration	node being scanned	resulting queue	label vector $(d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10})$
0		$Q = (0)$	$(0, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty)$
1	0	$Q = (1, 2)$	$(0, 1, 37, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty)$
2	1	$Q = (3, 4, 7, 2)$	$(0, 1, 2, 19, 35, \infty, \infty, 36, \infty, \infty, \infty)$
3	3	$Q = (5, 6, 8, 4, 7, 2)$	$(0, 1, 2, 19, 20, 27, 33, 36, 34, \infty, \infty)$
4	5	$Q = (10, 9, 6, 8, 4, 7, 2)$	$(0, 1, 2, 19, 20, 27, 28, 36, 34, 32, 30)$
5	10	$Q = (9, 6, 8, 4, 7, 2)$	$(0, 1, 2, 19, 20, 27, 28, 36, 34, 32, 30)$
6	9	$Q = (6, 8, 4, 7, 2)$	$(0, 1, 2, 19, 20, 27, 28, 36, 34, 32, 30)$
7	6	$Q = (10, 8, 4, 7, 2)$	$(0, 1, 2, 19, 20, 27, 28, 36, 34, 32, 29)$
8	10	$Q = (8, 4, 7, 2)$	$(0, 1, 2, 19, 20, 27, 28, 36, 34, 32, 29)$
9	8	$Q = (4, 7, 2)$	$(0, 1, 2, 19, 20, 27, 28, 36, 34, 32, 29)$
10	4	$Q = (5, 6, 7, 2)$	$(0, 1, 2, 19, 20, 21, 27, 36, 34, 32, 29)$
11	5	$Q = (10, 9, 6, 7, 2)$	$(0, 1, 2, 19, 20, 21, 22, 36, 34, 26, 24)$
12	10	$Q = (9, 6, 7, 2)$	$(0, 1, 2, 19, 20, 21, 22, 36, 34, 26, 24)$
13	9	$Q = (6, 7, 2)$	$(0, 1, 2, 19, 20, 21, 22, 36, 34, 26, 24)$
14	6	$Q = (10, 7, 2)$	$(0, 1, 2, 19, 20, 21, 22, 36, 34, 26, 23)$
15	10	$Q = (7, 2)$	$(0, 1, 2, 19, 20, 21, 22, 36, 34, 26, 23)$
16	7	$Q = (2)$	$(0, 1, 2, 19, 20, 21, 22, 36, 34, 26, 23)$
17	2	$Q = (3, 4)$	$(0, 1, 2, 3, 19, 21, 22, 36, 34, 26, 23)$
18	3	$Q = (5, 6, 8, 4)$	$(0, 1, 2, 3, 4, 11, 17, 36, 18, 26, 23)$
19	5	$Q = (10, 9, 6, 8, 4)$	$(0, 1, 2, 3, 4, 11, 12, 36, 18, 16, 14)$
20	10	$Q = (9, 6, 8, 4)$	$(0, 1, 2, 3, 4, 11, 12, 36, 18, 16, 14)$
21	9	$Q = (6, 8, 4)$	$(0, 1, 2, 3, 4, 11, 12, 36, 18, 16, 14)$
22	6	$Q = (10, 8, 4)$	$(0, 1, 2, 3, 4, 11, 12, 36, 18, 16, 13)$
23	10	$Q = (8, 4)$	$(0, 1, 2, 3, 4, 11, 12, 36, 18, 16, 13)$
24	8	$Q = (4)$	$(0, 1, 2, 3, 4, 11, 12, 36, 18, 16, 13)$
25	4	$Q = (5, 6)$	$(0, 1, 2, 3, 4, 5, 11, 36, 18, 16, 13)$
26	5	$Q = (10, 9, 6)$	$(0, 1, 2, 3, 4, 5, 6, 36, 18, 10, 8)$
27	10	$Q = (9, 6)$	$(0, 1, 2, 3, 4, 5, 6, 36, 18, 10, 8)$
28	9	$Q = (6)$	$(0, 1, 2, 3, 4, 5, 6, 36, 18, 10, 8)$
29	6	$Q = (10)$	$(0, 1, 2, 3, 4, 5, 6, 36, 18, 10, 7)$

Table 1: Iterations for the example network with $|N| = 11$

Arc	**Length**	**Arc** (for $k = 1, 2, ..., m-1$)	**Length**
$(0, 1)$	1	$(2k-1, 2m+k)$	$10 \times 2^{m-k} - 5$
$(0, 2)$	$10 \times 2^{m-1} - 3$	$(2k-1, 2k+2)$	$10 \times 2^{m-k} - 6$
$(2m-1, 3m)$	5	$(2k-1, 2k+1)$	$10 \times 2^{m-k-1} - 2$
$(2m-1, 3m+1)$	3	$(2k-1, 2k)$	1
$(2m-1, 2m)$	1	$(2k, 2k+2)$	$10 \times 2^{m-k-1} - 3$
$(2m, 3m+1)$	1	$(2k, 2k+1)$	1

Table 2: Arc lengths for the example network with $|N| = 3m + 2$

label vector represent the lengths of the shortest possible paths from node 0 to all other nodes that have been examined by the algorithm by the time when the corresponding queue is generated. The following results are proved mainly by observing these queues and the corresponding labels of nodes.

Lemma 1. (1) For any k, $1 \leq k \leq m$, any queue with top node $2k-1$, $Q = (2k-1, ...)$ contains node $2k$ as its second top node (hence actually $Q = (2k-1, 2k, ...)$) and does not contain any node in set $R_k \equiv \{2j-1, 2j | k+1 \leq j \leq m\} \cup \{2m+j | k \leq j \leq m\}$, and if the top node in the queue, $2k-1$, is scanned, its successors $2m+k$, $2k+2$, $2k+1$ will be added to the top of the queue respectively (hence the resulting queue will be $\bar{Q} = (2k+1, 2k+2, 2m+k, 2k, ...)$);

(2) For any k, $1 \leq k \leq m$, any queue with top node $2k$, $Q = (2k, ...)$ does not contain any node in set R_k, and if the top node in the queue, $2k$, is scanned, its successors $2k+2$, $2k+1$ will be added to the top of the queue (hence the resulting queue will be $\bar{Q} = (2k+1, 2k+2, ...)$).

Proof: See Chen and Powell [4]. □

Theorem 2. Upon termination of the algorithm, both node $2k-1$ and node $2k$

$(k = 1, 2, ..., m)$ are scanned exactly 2^{k-1} times.

Proof: We prove it by induction. For $k = 1, 2, 3$, it can be easily verified from Table 1. Now suppose it holds when $k = h$ for some $h \geq 4$. We need to show that it holds when $k = h + 1$. Suppose the current queue is $Q_0 = (2h - 1, ...)$ with top node $2h - 1$. By Lemma 1, node $2h$ is the second node in Q_0, i.e. $Q_0 = (2h - 1, 2h, ...)$. Now remove node $2h - 1$ from Q_0 and scan it. Suppose this is the j-th time $(1 \leq j \leq 2^{h-1})$ node $2h - 1$ gets scanned. By Lemma 1, right after $2h - 1$ is scanned the resulting queue is $Q_1 = (2h+1, 2h+2, 2m+h, 2h, ...)$. Before $2h-1$ can be scanned for the $(j+1)$-th time, nodes $2h + 1, 2h + 2, 2m + h, 2h$ in Q_1 and possibly their successors must be removed and scanned. By Lemma 1, right after node $2h$ in Q_1 is scanned the resulting queue becomes $Q_2 = (2h + 1, 2h + 2, ...)$. It is easy to see that both node $2h + 1$ and $2h + 2$ are scanned exactly once during the time from Q_1 to Q_2. There are two cases:

Case 1: If $j < 2^{h-1}$, then there must exist some node in set $R \equiv \{2i - 1, 2i | 1 \leq i \leq h - 1\}$ that is contained in Q_2 because otherwise it is impossible for node $2h - 1$ to get scanned for the $(j + 1)$-th time, violating the induction assumption that node $2h - 1$ is scanned 2^{h-1} times. Let $q \in \{2p - 1, 2p\}$ (with $1 \leq p \leq h - 1$) be the earliest node in Q_2 that belongs to the set R. Hence Q_2 can be written as $(2h + 1, 2h + 2, U, q, ...)$ where U is some subset of nodes. By Lemma 1, $(U \cup \{q\}) \cap R_{h+1} = \phi$ (where R_{h+1} is defined in Lemma 1), thus $U \subseteq \{2m + i | 1 \leq i \leq h\}$. Hence starting from Q_2 the algorithm scans both node $2h + 1$ and $2h + 2$ exactly once when it generates queue $Q_3 = (q, ...)$. By Lemma 1, it can be easily shown that starting from Q_3 the algorithm generates a sequence of queues $Q_3^i = (2p+i, 2p+i+1, ...)$ $(i = 1, 2, ...)$, and finally queue $Q_4 = (2h - 1, 2h, ...)$. Clearly, during this process, neither node $2h + 1$ nor node $2h + 2$ is scanned. Therefore, during the time period from Q_1, i.e. the time right after node $2h - 1$ is scanned for the j-th time, to Q_4, i.e. the time right before node $2h - 1$ can be scanned for the $(j + 1)$-th time, both node $2h + 1$ and $2h + 2$ are scanned exactly twice.

Case 2: If $j = 2^{h-1}$, then there is no node in R that is contained in Q_2 because otherwise node j will be scanned more than 2^{h-1} times, violating the induction assumption.

By Lemma 1, no node in Q_2 belongs to set R_{h+1}, thus except nodes $2h+1$ and $2h+2$, all the nodes in Q_2 belong to $\{2m+i | 1 \leq i \leq h\}$. Hence starting from Q_2 the algorithm scans both $2h+1$ and $2h+2$ exactly once when it terminates. Therefore, both $2h+1$ and $2h+2$ are scanned exactly twice during the time period from the time right after the j-th scanning of node $2h-1$ to the time when the algorithm terminates.

By the facts shown in Case 1 and Case 2 and the induction assumption that upon termination of the algorithm node $2h-1$ is scanned exactly 2^{h-1} times, we have shown that upon termination of the algorithm both node $2h+1$ and $2h+2$ are scanned exactly 2^h times. Thus, by induction, we have proved the theorem. □

This exponential worst-case situation is caused not only by the unusual arc lengths but also by the order of the successors examined while scanning a node. In Bertsekas' **SLF** strategy, this order is not specified. In fact, this order plays an important role in the worst case performance of the strategy. In the above example, if $j \in \Gamma(i)$ are examined in a nondecreasing order of d_{ij}, instead of a nonincreasing order, then the example can be solved in polynomial time. We show in the next section that the worst-case complexity of the **SLF** strategy becomes polynomial if a proper order is selected to examine the successors when scanning a node.

Bertsekas [2] suggests another slightly different version: **SLF**-threshold strategy by combining the **SLF** strategy and the threshold method of [5]. It should be noted that the worst case complexity of **SLF**-threshold strategy is the same as that of the **SLF** strategy because the **SLF**-threshold strategy is equivalent to the **SLF** strategy when the threshold is set sufficiently large.

2 A polynomial strategy

Bertsekas, Guerriero and Musmanno [3] give a modification of the **SLF** strategy that has a polynomial worst-case complexity with order $O(|N||A|)$. In this section, we give another modification that has a higher worst-case complexity but is in a much simpler manner than the one in [3].

We have noticed in Section 1 that the worst-case complexity of the **SLF** strategy is dependent on the order used to examine the successors when scanning a node. So a possible way of making the **SLF** strategy polynomial is to select this order properly rather than arbitrarily.

We suggest the following *shortest arc first* (**SAF**) strategy to specify this order:

Whenever scanning a node i, the successors $j \in \Gamma(i)$ are examined in the nondecreasing order of d_{ij}.

We call the resulting method **SLF-SAF** strategy which uses Bertsekas' **SLF** strategy whenever a node enters the queue Q, and uses the above **SAF** strategy whenever a node is scanned.

Recall that we assume each d_{ij} is nonnegative. In the following, we show that under this assumption the worst case complexity of the **SLF-SAF** strategy is bounded by $O(|N|^2|A|)$.

Consider the current queue generated by the strategy **SLF-SAF**. Denote this queue by Q_1. Without loss of generality, suppose $Q_1 = (1, 2, ..., k)$ with top node 1 and bottom node k. As the algorithm continues, the top node of Q_1, node 1, is removed, scanned, and some nodes of $N \setminus Q_1$ may be inserted in front of node 2 and some other nodes of $N \setminus Q_1$ may be inserted behind node k, creating a new queue $(..., 2, 3, ..., k, ...)$. Now, the top node of this queue is removed and scanned and the queue is updated again. At some iteration, we can have a queue $Q_2 = (2, 3, ..., k, ...)$ which is the first queue with top node 2 since scanning node 1 of Q_1. Similarly, we can have queues $Q_3 = (3, 4, ..., k, ...), ..., Q_i = (i, i + 1, ..., k, ...), ..., Q_k = (k, ...)$, where Q_i is the first queue with top node i since scanning node 1 of Q_1. Denote the queue immediately after scanning node k of Q_k by $\bar{Q}_k$. Note that, given Q_1, the algorithm uniquely determines $Q_2, Q_3, ..., Q_k, \bar{Q}_k$. We focus on these queues in the following.

Lemma 3. The **SLF-SAF** strategy needs at most $|N \setminus Q_i|$ times of node scanning procedure to go from Q_i to Q_{i+1}.

Proof: Starting with Q_i, the algorithm removes node i, the top node of Q_i, and scans node i. If none of $\Gamma(i)$ is eligible to be put in front of $i+1$, then Q_{i+1} is generated immediately after scanning node i. Suppose some of $\Gamma(i)$ are inserted in front of $i+1$ after scanning node i. Let the resulting queue be $Q = (j_1, j_2, ..., j_h, i+1, ..., k, ...)$, where, obviously, $j_u \in \Gamma(i)$ for $u = 1, 2, ..., h$. Since we are using the **SLF** strategy, we have: $d_{j_1} \leq d_{j_2} \leq ... \leq d_{j_h}$, where d_v denotes the label of node v corresponding to queue Q. We also have: $d_{j_1} \geq d_{j_2} \geq ... \geq d_{j_h}$ since we are using the **SAF** strategy as well. Thus, $d_{j_1} = d_{j_2} = ... = d_{j_h}$. Now, remove and scan j_1. If some nodes of $\Gamma(j_1)$ are inserted in front of j_2, then the labels of those nodes will be equal to d_{j_h} as well since the arc lengths are nonnegative. We can conclude that if we start with queue Q_i, before queue Q_{i+1} is generated the labels of the nodes once inserted in front of node $i+1$ are all the same. Thus, if we start with queue Q_i, before queue Q_{i+1} is generated a node can appear in front of node $i+1$ at most once, which implies that this node will be scanned at most once before Q_{i+1} is generated. Since only the nodes in $N \setminus Q_i$ are eligible to be inserted in front of node $i+1$, the total number of nodes to be inserted in front of node $i+1$ is at most $|N \setminus Q_i|$. Hence before Q_{i+1} is generated, the algorithm will do at most $|N \setminus Q_i|$ times of the node scanning procedure. This ends the proof. □

Corollary 4. The **SLF-SAF** strategy needs at most O($|A|$) basic arithmetic operations to go from Q_i to Q_{i+1}.

Proof: We have shown in the proof of Lemma 3 that to go from Q_i to Q_{i+1}, the algorithm will scan at most $|N \setminus Q_i|$ nodes and each node is scanned at most once, thus each arc is examined at most once. So, the algorithm needs at most O($|A|$) basic arithmetic operations to go from Q_i to Q_{i+1}. □

Theorem 5. The worst-case time complexity of the **SLF-SAF** strategy is bounded by $O(|N|^2|A|)$.

Proof: Consider queue Q_1. Recall that $Q_1 = (1, 2, ..., k)$ as assumed earlier. Suppose node p has the smallest label among all the nodes of Q_1, i.e., $d_p = \min_{j \in Q_1}\{d_j\}$, where d_i denotes the label of node i corresponding to queue Q_1. Clearly, node p will never enter any queue after it is scanned since the arc lengths are nonnegative. Starting with Q_1, the algorithm will scan each node of Q_1 at least once before $\bar{Q}_k$ is generated. So node p will not appear in $\bar{Q}_k$ and any subsequent queue since then on. By Corollary 4, it needs at most $O(k|A|) \leq O(|N||A|)$ basic arithmetic operations to go from Q_1 to $\bar{Q}_k$. Now starting with queue $\bar{Q}_k$, similarly, after at most $O(|N||A|)$ basic arithmetic operations, some node $q \in N \setminus \{p\}$ will never appear in any subsequent queue. Since there are $|N|$ nodes, we can conclude that after at most $O(|N|^2|A|)$ basic arithmetic operations, no node will appear in the queue, i.e., the algorithm will terminate. This ends the proof. $\square$

3 Conclusion

This paper shows that the **SLF** algorithm is nonpolynomial, but can be made polynomial with a minor change. While this is an interesting theoretical result, the question always arises: What is the practical impact? In the worst case, the preprocessing phase of the **SLF-SAF** strategy needs $O(|N|)$ time to sort the arcs leaving a node. Does this additional sorting produce a practical benefit? Alternatively, it is possible to speculate that in practice, our algorithm might even be slower.

Using Bertsekas' network generator, we ran an extensive set of comparisons of the two algorithms on the same set of problems used in [2]. The results showed no practical difference between the algorithms. Of course, such experimental results always carry the qualification that they are limited by the nature of the network generator. However, at this time, we feel the results of this paper should be viewed in light of their theoretical interest, and not as a practical enhancement.

Appendix

Proof of Lemma 1: We prove Lemmma 1 by induction. For $k = 1, 2, 3$, both (1) and (2) can be verified from Table 1. Given any $h \geq 3$, let us assume that both (1) and (2) hold for each k with $3 \leq k \leq h$. We need to prove that both (1) and (2) hold for $k = h + 1$.

First we want to show that for any queue Q with top node $2h - 1$ or $2h$, the corresponding labels of nodes $2h+1$ and $2h+2$ satisfy: $d_{2h+2} = d_{2h+1}+1$ or $d_{2h+1} = d_{2h+2} = \infty$. It is proved as follows.

Given a queue Q with top node $2h - 1$ or $2h$, if neither node $2h - 1$ nor node $2h$ has been scanned once by the time Q is generated, then $d_{2h+1} = d_{2h+2} = \infty$ because the labels of $2h + 1$ and $2h + 2$ can be updated only when node $2h - 1$ or $2h$ is scanned. Otherwise, at least one of nodes $2h - 1$, $2h$ has been scanned at least once by the time Q is generated. In this case, we claim that node $2h - 1$ must have been scanned at least once. The reason is as follows. If node $2h$ has been scanned at least once before Q was generated, then suppose that right before $2h$ was scanned last time before Q was generated the queue was $\bar{Q} = (2h, ...)$. It is easy to see that before this queue $\bar{Q}$ was generated, there was a node $j \in \{2h-1, 2h-2, 2h-3\}$ that must have been scanned at least once because $2h-1, 2h-2, 2h-3$ are the predecessors of $2h$. If $j = 2h-1$, then we have proved the claim. If $j = 2h - 2$ or $2h - 3$, then $\bar{Q}$ was generated by the algorithm starting with a queue $\tilde{Q} = (2h - 3, ...)$ or $\tilde{Q} = (2h - 2, ...)$. Removing and scanning the top node of $\tilde{Q}$, by the induction assumption, the algorithm generated another queue $\hat{Q} = (2h - 1, 2h, ...)$. Obviously, to get to $\bar{Q}$ from $\hat{Q}$, the algorithm had to scan node $2h - 1$ in $\hat{Q}$ for at least once. This shows the claim.

So we need only to consider the case where node $2h - 1$ has been scanned at least once by the time Q is generated. Suppose that right before node $2h-1$ was scanned last time before Q was generated the queue was $Q_0 = (2h-1, ...)$. By the induction assumption, $2h$ was the second node in Q_0, i.e. $Q_0 = (2h-1, 2h, ...)$. Suppose the label of node $2h-1$ corresponding to Q_0 was $d^0_{2h-1} = y$. Starting with Q_0, the algorithm removed and scanned $2h-1$, by the induction assumption, resulting in a new queue $Q_1 = (2h+1, 2h+2, 2m+h, 2h...)$ with the corresponding labels of nodes $2h+1$ and $2h+2$ being $d^1_{2h+1} = y + 10 \times 2^{m-h-1} - 2$ and $d^1_{2h+2} = y = 10 \times 2^{m-h} - 6$ respectively. There are two cases.

Case 1: If $2h$ is the top node in queue Q, then Q was generated right after node $2m+h$ in Q_1 was removed and scanned. In this case, before Q was generated, the algorithm would have done the following. Starting with Q_1, the algorithm removed and scanned node $2h+1$, resulting in a new queue $Q_2 = (S, 2h+2, 2m+h, 2h, ...)$ where set S was a subset of $\{2h+3, 2h+4, 2m+h+1\}$. It is easy to see that the labels of $2h+1$ and $2h+2$ corresponding to Q_2 became $d^2_{2h+1} = d^1_{2h+1} = y + 10 \times 2^{m-h-1} - 2$ and $d^2_{2h+2} = d^2_{2h+1} + 1 = y + 10 \times 2^{m-h-1} - 1$ respectively. To get to Q, the algorithm proceeded starting with Q_2 by removing and scanning the nodes $S \cup \{2h+2, 2m+h\}$ and possibly their successors. This process did not improve the labels of nodes $2h+1$ and $2h+2$ since nodes $S \cup \{2h+2, 2m+h\}$ and their successors do not have any arc connecting with $2h+1$ or $2h+2$. Thus when Q is generated now, the labels of $2h+1$ and $2h+2$ stay the same since Q_2 was generated. Hence the labels of $2h+1$ and $2h+2$ corresponding to Q are $d_{2h+1} = d^2_{2h+1}$ and $d_{2h+2} = d^2_{2h+2}$ respectively. This shows that $d_{2h+2} = d_{2h+1} + 1$ since $d^2_{2h+2} = d^2_{2h+1} + 1$.

Case 2: If $2h-1$ is the top node in queue Q, then Q was generated after node $2h$ in Q_1 was removed and scanned. In this case, before Q was generated, the algorithm had done the following. First, exactly as in Case 1, starting with Q_1, the algorithm removed and scanned node $2h+1$ and later nodes $S \cup \{2h+2, 2m+h\}$ and possibly some of their successors, resulting in a new queue $Q_3 = (2h, ...)$ with top node $2h$. Note that in Case 1, this Q_3 is exactly Q. But now since Q is with the top node $2h-1$ (instead of $2h$), in order to get to Q the algorithm had to proceed starting with Q_3. Suppose the label of

$2h$ corresponding to Q_3 is z. Removing and scanning node $2h$ in Q_3, by the induction assumption, the algorithm generated a new queue $Q_4 = (2h+1, 2h+2, ...)$ with the corresponding labels of $2h+1$ and $2h+2$ being $d^4_{2h+1} = z+1$ and $d^4_{2h+2} = z+10\times 2^{m-h-1}-3$ respectively. Then node $2h+1$ was removed from Q_4 and scanned, resulting in a new queue $Q_5 = (T, 2h+2, ...)$ where T was a subset of $\{2h+3, 2h+4, 2m+h+1\}$. The labels of $2h+1$ and $2h+2$ with respect to Q_5 became $d^5_{2h+1} = d^4_{2h+1}$ and $d^5_{2h+2} = d^4_{2h+1} + 1$ respectively. It must be true that Q_5 contained some node $j \in \{2i-1, 2i | 1 \leq i \leq h-1\}$ because otherwise it was impossible for the algorithm to get to Q starting from Q_5. Let node $q = 2p-1$ or $2p$ (with $p < h$) be the earliest (leftmost) node in Q_5 that belongs to the set $\{2i-1, 2i | 1 \leq i \leq h-1\}$. Without loss of generality, suppose $Q_5 = (T, 2h+2, U, q, V)$ where U and V were some subsets of nodes. Then U consisted of a subset of nodes in $\{2h+3, 2h+4, ..., 2m\} \cup \{2m+h+2, ..., 3m+1\}$. Obviously, scanning nodes in $T \cup \{2h+2\} \cup U$ and their successors did not affect the labels of $2h+1$ and $2h+2$. Thus when the algorithm, starting from Q_5, got to queue $Q_6 = (q, V)$ with top node q, the labels of $2h+1$ and $2h+2$ were not changed. Starting with Q_6, by the induction assumption, the algorithm generated a sequence of queues, one following another, with the top two nodes being $2p+1$ and $2p+2$; $2p+3$ and $2p+4$; ..., respectively, and eventually, it generated the queue Q with top two nodes $2h-1$ and $2h$. Clearly, during this process, the labels of nodes $2h+1$ and $2h+2$ were not improved. Therefore, from the time after Q_5 was generated to the time when Q is generated now, the labels of $2h+1$ and $2h+2$ kept unchanged. Thus the labels of $2h+1$ and $2h+2$ corresponding to Q are: $d_{2h+2} = d^5_{2h+2} = d^5_{2h+1} + 1$ and $d_{2h+1} = d^5_{2h+1}$ respectively. Hence $d_{2h+2} = d_{2h+1} + 1$.

Combining Case 1 and Case 2, we have shown that for any queue Q with top node $2h-1$ or $2h$, the corresponding labels of nodes $2h+1$ and $2h+2$ satisfy: $d_{2h+2} = d_{2h+1}+1$ or $d_{2h+1} = d_{2h+2} = \infty$.

In the following, we first prove that (1) of Lemma 1 holds when $k = h+1$, then prove that (2) of Lemma 1 also holds when $k = h+1$.

Part (1) First let us see what the labels of nodes $2h+1$, $2h+2$, $2m+h$, $2h+3$, $2h+4$ and $2m+h+1$ will be with respect to a queue $Q^0 = (2h+1, ...)$ with top node

$2h+1$, and then show that scanning the top node $2h+1$ of Q^0 will add nodes $2m+h+1$, $2h+4$ and $2h+3$ to the top positions of the resulting queue respectively and hence show the correctness of (1) of Lemma 1 when $k = h+1$. By the induction assumption, whenever node $2h-1$ or $2h$ is scanned, node $2h+1$ becomes the top node in the resulting queue. On the other hand, nodes $2h-1$ and $2h$ are the only predecessors of node $2h+1$, therefore queue Q^0 must be generated right after node $2h-1$ or $2h$ is scanned. Let $Q^- = (j, p, \ldots)$ be the queue right before node $2h-1$ (or $2h$) is scanned where $j = 2h-1$ (or $2h$). Then Q^0 is generated right after removing node j from Q^- and then scanning it. By the result we proved earlier that with respect to Q^-, the labels of nodes $2h+1$ and $2h+2$, d^-_{2h+1} and d^-_{2h+2} satisfy:

$$d^-_{2h+2} = d^-_{2h+1} + 1 \quad \text{or} \quad d^-_{2h+1} = d^-_{2h+2} = \infty.$$

By the network structure, it is easy to show that with respect to Q^-, the labels of nodes $2h+3$, $2h+4$ and $2m+h+1$, d^-_{2h+3}, d^-_{2h+4} and d^-_{2m+h+1} satisfy:

$$\begin{aligned} d^-_{2h+3} &\geq d^-_{2h+1} + 2 \\ d^-_{2h+4} &\geq d^-_{2h+1} + 3 \\ d^-_{2m+h+1} &\geq d^-_{2h+1} + 10 \times 2^{m-h-1} - 5. \end{aligned} \tag{1}$$

There are two cases with respect to the top node of Q^-:

Case 1: The top node of Q^-, $j = 2h-1$. By the induction assumption, removing and scanning node $2h-1$ in Q^- results in $Q^0 = (2h+1, 2h+2, 2m+h, p, \ldots)$, which implies that the labels of nodes $2h+1, 2h+2, 2m+h$ are improved such that their new labels corresponding to Q^0, d^0_{2h+1}, d^0_{2h+2} and d^0_{2m+h} satisfy:

$$d^0_{2h+1} = d^-_{2h-1} + 10 \times 2^{m-h-1} - 2 < d^-_{2h+1} \tag{2}$$

$$d^0_{2h+2} = d^-_{2h-1} + 10 \times 2^{m-h} - 6 < d^-_{2h+2} = d^-_{2h+1} + 1 \tag{3}$$

$$d^0_{2m+h} = d^-_{2h-1} + 10 \times 2^{m-h} - 5. \tag{4}$$

where d^-_{2h-1} is the label of node $2h-1$ corresponding to queue Q^-. While the labels of nodes $2m+h+1$, $2h+3$ and $2h+4$ are not changed, i.e. their labels corresponding to Q^0, d^0_{2m+h+1}, d^0_{2h+3} and d^0_{2h+4} satisfy:

$$d^0_{2m+h+1} = d^-_{2m+h+1} \tag{5}$$

$$d^0_{2h+3} = d^-_{2h+3} \geq d^-_{2h+1} + 2 \tag{6}$$

$$d^0_{2h+4} = d^-_{2h+4} \geq d^-_{2h+1} + 3 \tag{7}$$

The relations of labels of nodes $2h+1, 2h+2, 2m+h, 2h+3, 2h+4, 2m+h+1$ corresponding to queue Q^0 are implied in equations (2)-(7). Recall that $Q^0 = (2h+1, 2h+2, 2m+h, p, ...)$. Now we want to show that scanning the top node $2h+1$ of Q^0 will add nodes $2m+h+1$, $2h+4$ and $2h+3$ to the top positions of the resulting queue respectively. Remove $2h+1$ from Q^0 and scan it. If the labels of nodes $2h+3, 2h+4, 2m+h+1$ are improved after node $2h+1$ is scanned, their new labels should be:

$$d^1_{2h+3} = d^0_{2h+1} + 10 \times 2^{m-h-2} - 2 \tag{8}$$

$$d^1_{2h+4} = d^0_{2h+1} + 10 \times 2^{m-h-1} - 6 \tag{9}$$

$$d^1_{2m+h+1} = d^0_{2h+1} + 10 \times 2^{m-h-1} - 5 \tag{10}$$

respectively. Now let us show that they are indeed improved, i.e. $d^1_i < d^0_i$ for $i = 2h+3, 2h+4, 2m+h+1$. By (6) and (3), we have

$$d^0_{2h+3} > d^-_{2h-1} + 10 \times 2^{m-h} - 5 \tag{11}$$

Thus by (8) and (2), it follows that

$$\begin{aligned} d^1_{2h+3} &= d^-_{2h-1} + 10 \times 2^{m-h-1} - 2 + 10 \times 2^{m-h-2} - 2 \\ &< d^-_{2h-1} + 10 \times 2^{m-h} - 5 < d^0_{2h+3} \end{aligned} \tag{12}$$

Similarly, by (7) and (3), we have:

$$d^0_{2h+4} > d^-_{2h-1} + 10 \times 2^{m-h} - 4, \tag{13}$$

and by (9) and (2), we can prove that $d^1_{2h+4} < d^0_{2h+4}$. Finally, by (1), (2), (5) and (10), we can easily prove that $d^1_{2m+h+1} < d^0_{2m+h+1}$.

By the induction assumption that none of nodes in set R_h is contained in Q^- and the fact that during the process from Q^- to Q^0, none of these nodes is added to Q^0 except $2h+1$, $2h+2$ and $2m+h$, thus Q^0 does not contain any node in R_{h+1}. Particularly, Q^0 does not contain any node in $\{2h+3, 2h+4, 2m+h+1\}$. Thus nodes $2h+3, 2h+4, 2m+h+1$ are eligible to enter the queue when scanning the top node $2h-1$ of Q^0. By (10) and (2), we have: $d^1_{2m+h+1} = d^-_{2h-1} + 10 \times 2^{m-h} - 7$. Hence by (3), $d^1_{2m+h+1} < d^0_{2h+2}$. On the other hand, by (8), (9) and (10, we have: $d^1_{2m+h+1} > d^1_{2h+4} > d^1_{2h+3}$. Thus when scanning the top node $2h+1$ of Q^0, the algorithm, which examines the successors of node $2h+1$ in the nonincreasing order of their distances from $2h+1$, must add nodes $2m+h+1$, $2h+4$ and $2h+3$ to the top positions of the resulting queue respectively. This shows the correctness of (1) of Lemma 1 when $k = h+1$.

So, right after node $2h+1$ is scanned, the resulting queue will be $Q^1 = (2h+3, 2h+4, 2m+h+1, 2h+2, 2m+h, p, \ldots)$. The corresponding labels of the nodes $2h+3, 2h+4, 2m+h+1$ will be $d^1_{2h+3}, d^1_{2h+4}, d^1_{2m+h+1}$ as shown in (8), (9) and (10) respectively. It is easy to check that $d^0_{2h+2} > d^0_{2h+1} + d_{2h-1,2h+2}$, thus during the process from Q^0 to Q^1, the label of node $2h+2$ is improved and its new label with respect to Q^1, d^1_{2h+2} satisfies:

$$d^1_{2h+2} = d^0_{2h+1} + 1. \tag{14}$$

While the labels of nodes $2h+1, 2m+h$ are not improved, i.e. they are $d^1_{2h+1} = d^0_{2h+1}$ and $d^1_{2m+h} = d^0_{2m+h}$ respectively. By (4) and (2), d^1_{2m+h} can be rewritten as:

$$d^1_{2m+h} = d^0_{2h+1} + 10 \times 2^{m-h-1} - 3. \tag{15}$$

The labels d^1_{2h+2}, d^1_{2h+3}, d^1_{2h+4}, and d^1_{2m+h} characterized here are used later to prove some result.

Case 2: The top node of Q^- is $j = 2h$. By the induction assumption, removing and scanning node $2h$ in Q^- results in $Q^0 = (2h+1, 2h+2, p, ...)$, which implies that the labels of nodes $2h+1, 2h+2$ are improved such that their new labels corresponding to Q^0 are:

$$d^0_{2h+1} = d^-_{2h} + 1 < d^-_{2h+1} \tag{16}$$

$$d^0_{2h+2} = d^-_{2h} + 10 \times 2^{m-h-1} - 3 < d^-_{2h+2} = d^-_{2h+1} + 1 \tag{17}$$

respectively, where d^-_{2h} is the label of node $2h$ corresponding to queue Q^-. Since by the induction assumption node p is not in R_h, the label of node p is not improved during the process from Q^- to Q^0. On the other hand, nodes $2h+1$ and $2h+2$ are added to the top positions earlier than node p, thus it must be true that:

$$d^0_p = d^-_p > d^0_{2h+2} \tag{18}$$

where d^-_p and d^0_p are the labels of node p with respect to queue Q^- and Q^0 respectively. While the labels of nodes $2m+h+1$, $2h+3$ and $2h+4$ are not changed, i.e. their labels corresponding to Q^0 can be described by (5), (6) and (7) respectively. The relations of labels of nodes $2h+1, 2h+2, 2h+3, 2h+4, 2m+h+1, p$ corresponding to queue Q^0 are thus implied in equations (16)-(18) and (5)-(7). Recall that $Q^0 = (2h+1, 2h+2, p, ...)$. In the following we show that scanning the top node $2h+1$ of Q^0 will add nodes $2m+h+1$, $2h+4$ and $2h+3$ to the top positions of the resulting queues respectively. Remove $2h+1$ from Q^0 and scan it. If the labels of nodes $2h+3, 2h+4, 2m+h+1$ are improved after node $2h+1$ is scanned, their new labels should be given by (8), (9) and (10) respectively. As in Case 1, we can show that they are indeed improved as follows. By (6) and (17), we have

$$d^0_{2h+3} > d^-_{2h} + 10 \times 2^{m-h-1} - 2 \tag{19}$$

Thus by (8) and (16), it follows that

$$d^1_{2h+3} \quad = \quad d^-_{2h} + 10 \times 2^{m-h-2} - 1$$

$$< \quad d^{-}_{2h} + 10 \times 2^{m-h-1} - 2 < d^{0}_{2h+3} \tag{20}$$

Similarly, by (7) and (17), we have:

$$d^{0}_{2h+4} > d^{-}_{2h} + 10 \times 2^{m-h-1} - 1, \tag{21}$$

and by (9) and (16), we can prove that $d^{1}_{2h+4} < d^{0}_{2h+4}$. Finally, we can easily prove that $d^{1}_{2m+h+1} < d^{0}_{2m+h+1}$.

Using the same argument as in Case 1, we can show that Q^0 contains none of the nodes in R_{h+1}, and that nodes $2h+3, 2h+4, 2m+h+1$ are eligible to enter the queue when scanning the top node $2h+1$ of Q^0. By (10) and (16), we have: $d^{1}_{2m+h+1} = d^{-}_{2h} + 10 \times 2^{m-h-1} - 4$. Hence by (17), $d^{1}_{2m+h+1} < d^{0}_{2h+2}$. On the other hand, it is easy to show that $d^{1}_{2m+h+1} > d^{1}_{2h+4} > d^{1}_{2h+3}$. Thus when scanning the top node $2h$ of Q^0, as in Case 1, the algorithm will add nodes $2m+h+1$, $2h+4$ and $2h+3$ to the tops of the subsequent queues respectively. This shows the correctness of (1) of Lemma 1 when $k = h+1$.

So, right after Q^0 is scanned, the resulting queue will be $Q^1 = (2h+3, 2h+4, 2m+h+1, 2h+2, p, ...)$. The corresponding labels of the nodes $2h+3, 2h+4, 2m+h+1$ will be $d^{1}_{2h+3}, d^{1}_{2h+4}, d^{1}_{2m+h+1}$ as shown in (8), (9) and (10) respectively. As in Case 1, the label of node $2h+2$ with respect to Q^1 is improved and given by (14). It is easy to see that during this process from Q^0 to Q^1, the label of p is not improved. Hence by (18), (16) and (17), the label of p corresponding to Q^1, d^{1}_{p} can be written as:

$$\begin{aligned} d^{1}_{p} &= d^{0}_{p} > d^{0}_{2h+2} \\ &= d^{0}_{2h+1} + 10 \times 2^{m-h-1} - 4 \end{aligned} \tag{22}$$

<u>**Part (2)**</u> Now let us turn to a queue $\bar{Q}^0 = (2h+2, ...)$ with top node $2h+2$. We first characterize $\bar{Q}^0$ and then show that scanning the top node $2h+2$ of $\bar{Q}^0$ will add nodes $2h+3$ and $2h+4$ to the top positions of the resulting queues and hence show the correctness of (2) of Lemma 1 when $k = h+1$. By the induction assumption, whenever node $2h-1$ or $2h$ is scanned, nodes $2h+1$ and $2h+2$ become the top node and the second

top node respectively in the resulting queue. Thus queue $\bar{Q}^0$ must be generated some iterations after node $2h+1$ is scanned. Thus to characterize queue $\bar{Q}^0$, we need only to check the relevant queues generated by the algorithm starting with a queue Q^0 with top node $2h+1$. As we have shown in **Part (1)**, this Q^0 has the following properties: either as in Case 1 of **Part (1)**, i.e. $Q^0 = (2h+1, 2h+2, 2m+h, p, ...)$ with the labels of nodes $2h+1$, $2h+2$, $2m+h$, $2h+3$ and $2h+4$ being given by (2), (3), (4), (6) and (7) respectively, or as in Case 2 of **Part (1)**, i.e. $Q^0 = (2h+1, 2h+2, p, ...)$ with the labels of nodes $2h+1$, $2h+2$, p, $2h+3$ and $2h+4$ being given by (16), (17), (18), (6) and (7) respectively. So there are two possible cases.

Case 1. If $Q^0 = (2h+1, 2h+2, 2m+h, p, ...)$ as in Case 1 of **Part (1)**, then to get to queue $\bar{Q}^0 = (2h+2, ...)$, the algorithm first removes $2h+1$ from Q^0 and scan it. As shown in Case 1 of **Part (1)**, this results in queue $Q^1 = (2h+3, 2h+4, 2m+h+1, 2h+2, 2m+h, p, ...)$ with the corresponding labels of nodes $2h+3$, $2h+4$, $2m+h+1$, $2h+2$ and $2m+h$ being given by (8), (9), (10), (14) and (15) respectively. The algorithm proceeds by removing $2h+3$ from Q^1 and scanning it, which results in a new queue $Q^2 = (S, 2h+4, 2m+h+1, 2h+2, 2m+h, p, ...)$ where $S \subseteq \{2h+5, 2h+6, 2m+h+2\}$. It is easy to prove that during the process from Q^1 to Q^2, the label of node $2h+4$ gets improved and its new label is $d^2_{2h+4} = d^1_{2h+3} + 1$. But the labels of nodes $2h+3$, $2h+2$ and $2m+h$ are not changed. To get to $\bar{Q}^0$, the algorithm needs to remove and scan all the nodes (and possibly their successors) with positions earlier than $2h+2$ in Q^2. Clearly, this whole procedure will not improve the labels of nodes $2h+3$, $2h+4$, $2h+2$ and $2m+h$. Thus when eventually $\bar{Q}^0$ is generated, the labels of these nodes, $\bar{d}_{2h+3}$, $\bar{d}_{2h+4}$, $\bar{d}_{2h+2}$ and $\bar{d}_{2m+h}$ must saisfy:

$$\bar{d}_{2h+3} = d^1_{2h+3} \tag{23}$$

$$\bar{d}_{2h+4} = d^1_{2h+3} + 1 \tag{24}$$

$$\bar{d}_{2h+2} = d^1_{2h+2} \tag{25}$$

$$\bar{d}_{2m+h} = d^1_{2m+h} \tag{26}$$

where d^1_{2h+3}, d^1_{2h+2} and d^1_{2m+h} are given in (8), (14) and (15) respectively, and actually $\bar{Q}^0 = (2h+2, 2m+h, p, ...)$. By the induction assumption, it is easy to prove that no node in set R_{h+1} is contained in $\bar{Q}^0$. Particularly, neither $2h+3$ nor $2h+4$ is in $\bar{Q}^0$. Now remove $2h+2$ from $\bar{Q}^0$ and scan it. The labels of nodes $2h+3$ and $2h+4$ will be improved because by (23), (24), (25), (8) and (14), we have:

$$\begin{aligned} d_{2h+3} &= \bar{d}_{2h+2} + d_{2h+2,2h+3} \\ &= d^0_{2h+1} + 2 \\ &< d^0_{2h+1} + 10 \times 2^{m-h-2} - 2 = d^1_{2h+3} = \bar{d}_{2h+3}, \end{aligned} \tag{27}$$

and

$$\begin{aligned} d_{2h+4} &= \bar{d}_{2h+2} + d_{2h+2,2h+4} \\ &= d^0_{2h+1} + 10 \times 2^{m-h-2} - 2 \\ &< d^0_{2h+1} + 10 \times 2^{m-h-2} - 1 = d^1_{2h+3} + 1 = \bar{d}_{2h+4}. \end{aligned} \tag{28}$$

Thus the new labels of nodes $2h+3$ and $2h+4$, d_{2h+3} and d_{2h+4} are given by (27) and (28) respectively. Clearly, the label of node $2m+h$ is not improved during the process of scanning node $2h+2$ of $\bar{Q}^0$. By (26), (15) and (28), it is easy to see that $d_{2h+4} < \bar{d}_{2m+h}$. Therefore, nodes $2h+3$ and $2h+4$ are added to the first two top positions of the resulting queue right after node $2h+2$ is scanned. This shows that (2) of Lemma 1 holds when $k = h+1$.

Case 2. If $Q^0 = (2h+1, 2h+2, p, \ldots)$ as in Case 2 of **Part (1)**, then similarly to Case 1, we can show that starting from this Q^0 the algorithm eventually generates queue $\bar{Q}^0 = (2h+2, p, \ldots)$ with the labels of nodes $2h+3$, $2h+4$, $2h+2$ and p, $\bar{d}_{2h+3}$, $\bar{d}_{2h+4}$, $\bar{d}_{2h+2}$ and $\bar{d}_p$ saisfying (23), (24), (25) and

$$\bar{d}_p = d^1_p \tag{29}$$

respectively, where d^1_p is given in (22). By the induction assumption, it is easy to prove that no node in set R_{h+1} is contained in $\bar{Q}^0$. Particularly, neither $2h+3$ nor $2h+4$ is in $\bar{Q}^0$. Now remove $2h+2$ from $\bar{Q}^0$ and scan it. Using the same arguments as in Case 1, we can prove that the labels of nodes $2h+3$ and $2h+4$ are improved and given by (27) and (28) respectively. Clearly, the label of node p is not improved during the process of scanning node $2h+2$ of $\bar{Q}^0$. By (29), (22) and (28), it is easy to see that $d_{2h+4} < \bar{d}_p$. Therefore, nodes $2h+3$ and $2h+4$ are added to the first two top positions of the resulting queue right after node $2h+2$ is scanned. This shows that (2) of Lemma 1 holds when $k = h+1$.

In summary, we have shown that both (1) and (2) of Lemma 1 hold when $k = h+1$. Thus by induction, we have shown the correctness of Lemma 1. □

Acknowledgement

This research was supported in part by grant AFOSR-F49620-93-1-0098 from the Air Force Office of Scientific Research. The authors would also like to thank Dimitri Bertsekas for the use of his network generator for comparing the different variations on his test data set.

References

[1] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, 1957.

[2] D. Bertsekas. A simple and fast label correcting algorithm for shortest paths. *Networks*, 23:703–709, 1993.

[3] D. Bertsekas, F. Guerriero, and R. Musmanno. Parallel asynchronous label correcting methods for shortest paths. *Journal of Optimization Theory and Application*, (to appear), 1994.

[4] Z.-L. Chen and W. Powell. A note on Bertsekas' small-label-first strategy. Technical report, Department of Civil Engineering and Operations Research, Princeton University, 1996.

[5] F. Glover, D. Klingman, N. Phillips, and R. Schneider. New polynomial shortest path algorithms and their computational attributes. *Management Science*, 31:1106–1128, 1985.

[6] U. Pape. Implementation and efficiency of Moore algorithms for the shortest route problem. *Math. Programming*, 7:212–222, 1974.