Available online at www.sciencedirect.com

ScienceDirect

Journal of Discrete Algorithms 5 (2007) 408–421

JOURNAL OF DISCRETE ALGORITHMS

www.elsevier.com/locate/jda

A Zero-Space algorithm for Negative Cost Cycle Detection in networks

K. Subramani [1]

LDCSEE, West Virginia University, Morgantown, WV, USA

Available online 1 February 2007

Abstract

This paper is concerned with the problem of checking whether a network with positive and negative costs on its arcs contains a negative cost cycle. The Negative Cost Cycle Detection (NCCD) problem is one of the more fundamental problems in network design and finds applications in a number of domains ranging from Network Optimization and Operations Research to Constraint Programming and System Verification. As per the literature, approaches to this problem have been either Relaxation-based or Contraction-based. We introduce a fundamentally new approach for negative cost cycle detection; our approach, which we term as the *Stressing Algorithm*, is based on exploiting the connections between the NCCD problem and the problem of checking whether a system of difference constraints is feasible. The Stressing Algorithm is an incremental, comparison-based procedure which is as efficient as the fastest known comparison-based algorithm for this problem. In particular, on a network with n vertices and m edges, the Stressing Algorithm takes $O(m \cdot n)$ time to detect the presence of a negative cost cycle or to report that none exists. *A very important feature of the Stressing Algorithm is that it uses zero extra space; this is in marked contrast to all known algorithms that require $\Omega(n)$ extra space*. It is well known that the NCCD problem is closely related to the Single Source Shortest Paths (SSSP) problem, i.e., the problem of determining the shortest path distances of all the vertices in a network, from a specified source; indeed most algorithms in the literature for the NCCD problem are modifications of approaches to the SSSP problem. At this juncture, it is not clear whether the Stressing Algorithm could be extended to solve the SSSP problem, even if $O(n)$ extra space is available.

Keywords: Negative cost cycles; Polyhedra; Difference constraints; Stressing; Relaxing, zero-space; Shortest paths

1. Introduction

This paper introduces a new, strongly polynomial time algorithm for the problem of checking whether a network (directed graph) with positive and negative costs on its arcs has a negative cost cycle. Briefly, let $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \vec{\mathbf{c}} \rangle$ denote a network with vertex set $\mathbf{V} = \{v_1, v_2, \ldots, v_n\}$, edge set $\mathbf{E} = \{e_1, e_2, \ldots, e_m\}$ and cost function $\vec{\mathbf{c}} : \mathbf{E} \rightarrow \mathbf{Z}$ that assigns an integer to the edges. The goal of the Negative Cost Cycle Detection (NCCD) problem is to check whether there exists a simple cycle in this network whose edge weights sum up to a negative number. The NCCD problem is one of the more fundamental problems in Network Design, spanning the areas of Operations Research [1], Theoretical Computer Science [4] and Artificial Intelligence [6]. This problem also finds applications in areas such as Image

E-mail address: ksmani@csee.wvu.edu.

[1] This research was fully supported by the Air-Force Office of Scientific Research under Contract FA9550-06-1-0550.

1570-8667/$ – see front matter
doi:10.1016/j.jda.2006.12.002

Segmentation [5] and Real-Time scheduling [16]. Note that the NCCD problem is equivalent to checking whether a system of difference constraints is feasible [4].

Approaches to NCCD can be broadly categorized as relaxation-based [4] or contraction-based [18]. Depending on the heuristic used to select edges to be relaxed, a number of relaxation-based approaches are possible. Each of these approaches works well on selected classes of networks, while performing poorly on other classes. This paper discusses a new approach for NCCD called the Stressing Algorithm; this algorithm exploits the connections between networks and systems of difference constraints to achieve the goal of negative cycle detection. A curious feature of the Stressing Algorithm is that it can be implemented in *zero* extra space, assuming that the space in which the input is stored is writable. All algorithms for NCCD, that we know of, use at least $\Omega(n)$ extra space on an n-vertex network, to maintain distance labels from the source. At this juncture, it is not clear to us whether the Stressing Algorithm can be modified to solve the Single-Source Shortest Paths (SSSP) problem, even if $O(n)$ extra space is available.

The rest of this paper is organized as follows: In Section 2, we discuss the motivation for our work. Section 3 details a number of approaches to the problem of detecting negative cost cycles and determining shortest paths in a network. The Stressing Algorithm and an analysis of its running time are presented in Section 4. Important definitions and theorems which are used in the paper are stated in Section 5. The correctness of the Stressing Algorithm is proved in Section 6. Section 7 briefly describes the problems associated with computing Single-Source Shortest paths, using the Stressing Algorithm. In Section 8, we contrast the Stressing Algorithm with the Bellman–Ford algorithm, clearly delineating the important differences between the two. We conclude in Section 9, by summarizing our contributions and pointing out avenues for future research.

2. Motivation

Inasmuch as difference constraints occur in a wide variety of domains, it is hardly necessary to motivate the study of NCCD. Some of the important application areas of NCCD include Image Segmentation [5], Temporal Constraint Solving [6,13], Scheduling [11] and System Verification [2]. Typically, difference constraint systems are solved by finding the single-source shortest paths or detecting negative cost cycles in the associated constraint network. We also note that negative cost cycle detection is an integral subroutine in other network optimization problems such as the min-cost flow problem [12].

The literature widely contends that NCCD is in some sense, equivalent to the SSSP problem [7], in that an algorithm for one of them can be modified into an algorithm for the other, without additional resources. In this context, all algorithms for both problems require $\Omega(n)$ space and $\Omega(m \cdot n)$ time. We have conjectured that NCCD should be "easier" than SSSP, since the former is a decision problem, whereas the latter is a search problem [18].

While we have not been able to reduce the time bounds for NCCD, our algorithm does successfully and substantially reduce the space required for NCCD, with respect to all known algorithms for either NCCD or SSSP.

A secondary concern with respect to Relaxation-based methods is that they depend upon the fact that *every vertex in the network must be reachable from the source*. For instance, if none of the vertices of a negative cost cycle in a network is reachable from the source, then a label-correcting approach will not be able to detect the same, since after $O(m \cdot n)$ iterations, the labels corresponding to these vertices will all be set to ∞ and the condition checking for the presence of negative cost cycles fails. If a dummy source is introduced, with arcs of cost 0 to every vertex in the network, then label-correcting approaches will work correctly; however, in this case, the labels associated with the vertices of the network, *do not correspond to the Single-Source Shortest Paths from the actual source*, but to the shortest paths from the dummy source. So in essence, a label-correcting approach for SSSP must be run twice; the first time to detect negative cost cycles and the second time to compute the shortest paths, assuming that negative cost cycles do not exist. It is of course, well known that computing shortest paths in the presence of negative cost cycles is a strongly `NP-Hard` problem [8]. We remark that the Stressing Algorithm does not require that the input network be connected in any manner; indeed, the notion of a source does not exist, insofar as this algorithm is concerned.

From the practical perspective, the Stressing Algorithm can be easily distributed making it attractive to applications in Adhoc Networks. As we shall see later, a node has to merely monitor its edges to detect the presence of inconsistent constraints.

3. Related work

Approaches to NCCD can be broadly classified into the following two categories:

(a) Relaxation-based methods—Relaxation-based methods (also called Label-correcting methods) maintain an upper-bound for each vertex from the source, successively improving this bound till the true distance from the source is reached. The relaxation approach of the Bellman–Ford algorithm (BF) is one of the earliest and to date, asymptotically fastest algorithm for NCCD, if the cost function $\vec{\mathbf{c}}$ is completely arbitrary. [1] presents several heuristics which can reduce the number of steps that are executed by the naive BF algorithm; these heuristics include using a FIFO queue (BFFI), a predecessor array (BFPR), and both a FIFO queue and a predecessor array (BFFP). Another variation of the BF algorithm combines BF with the subtree disassembly cycle detection strategy [19] (BFCT). The Goldberg–Radzik algorithm [10] (GORC) is the fastest known empirical strategy for the Negative Cycle Detection problem and is primarily designed to calculate the Single Source shortest paths in the input network.
Within the framework of relaxation-based methods, two interesting paradigms are provided by scaling-based approaches and network simplex-based approaches. The scaling-based approach is predicated on the following fact: When the edge costs are small integers, the shortest path from the source to a vertex cannot exceed $n \cdot N$, where N is the largest cost (in magnitude) over all the edges of the network. The scaling approach also uses relaxation over small domains and runs in time $O(\sqrt{n}m \cdot \log N)$. The network simplex-based approach uses relaxation as a means to identify optimality of the current labels or indicate unboundedness [1].
Engineering aspects of the Shortest Paths problem in directed graphs are discussed at length in [9] and [3].

(b) Contraction-based methods—Contraction-based methods are predicated on the principle that contracting a vertex in a network preserves shortest paths and hence negative cost cycles, if any. A vertex, say v_i, is contracted in the following manner: Each edge of the form $v_k \rightsquigarrow v_i$ is combined with each edge of the form $v_i \rightsquigarrow v_j$ to create a new edge of the form $v_k \rightsquigarrow v_j$, with edge cost equal to the sum of the edge costs of $v_k \rightsquigarrow v_i$ and $v_i \rightsquigarrow v_j$. Redundant edges are identified and eliminated on the fly. Finally, the vertex v_i and all its accompanying edges are deleted from the network. It is clear that after at most n contractions, a negative cost cycle, if it exists, will be discovered. For a detailed proof of the correctness of this technique and its performance profile, see [17,18].

In this paper, we focus on a new approach that is neither relaxation-based nor contraction-based; the focus of our algorithm is merely the detection of a negative cost cycle; we currently are not aware of a technique that permits the extraction of such a cycle, if it exists. We remark that the Stressing Algorithm and the Contraction-based algorithms apparently contradict the assertion in [3], that every negative cost cycle detection algorithm is a combination of a shortest path algorithm and a cycle detection strategy. Finally, note that the Stressing Algorithm is unique, in that it can be implemented using zero extra space, i.e., no temporary variables in the form of distance labels are required.

4. The Stressing Algorithm

Algorithm 4.1 describes the details of the Stressing Algorithm; at the heart of this algorithm, is the STRESS() procedure (Algorithm 4.2), which is applied to every vertex in each iteration of the outermost **for** loop. For notational

Function DETECT-NEGATIVE-CYCLE ($\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \vec{\mathbf{c}} \rangle$)

```
1:  for (r = 1 to n − 1) do
2:    for (each vertex v_i ∈ V) do
3:      Let c_ij denote the cost of the lightest (in terms of cost) edge entering v_i.
4:      {If v_i has no incoming edge, c_ij = +∞.}
5:      if (c_ij < 0) then
6:        STRESS(v_i, c_ij)
7:      end if
8:    end for
9:  end for
10: if (∃e_ij ∈ E : c_ji < 0) then
11:   G contains a negative cost cycle.
12: else
13:   G does not contain a negative cost cycle.
14: end if
```

Algorithm 4.1. Negative Cost Cycle Detection through Stressing.

Function STRESS(v_i, c_{ij})

1: Subtract c_{ij} from the cost of each edge entering v_i.
2: Add c_{ij} to the cost of each edge leaving v_i.

Algorithm 4.2. The Stressing Algorithm.

convenience, we shall refer to this loop as **L**; further the ith iteration of **L** will be denoted by $\mathbf{L}_i$. At this juncture it is important to clarify a notational peculiarity. The vertices of the network are labeled as $\mathbf{V} = \{v_1, v_2, \ldots, v_n\}$, while an edge from vertex v_i to vertex v_j is labeled e_{ij}. The cost of edge e_{ij} is denoted by c_{ji}. This is because edge e_{ij} is used to represent the difference constraint $x_j - x_i \leqslant c_{ji}$ (cf. Section 6 after Theorem 6.1).

To stress a vertex v_i, we first determine the cost of the least cost edge into v_i; if this edge is e_{ij}, then as per our notational scheme, its cost is c_{ji}. We then subtract c_{ji} from the cost of each edge into v_i and add c_{ji} to the cost of each edge leaving v_i.

Some of the salient features of Algorithm 4.1 that bear mention are as follows:

(a) A vertex v_i is stressed by Algorithm 4.1, only if it has an incoming edge of negative cost.
(b) If a vertex v_i has no outgoing edge, then we assume that it has an outgoing edge e_{iz}, such that $c_{zi} = \infty$, where v_z is a specialized "sink" vertex, such that each vertex of **G** has an edge into it. Accordingly, if v_i is stressed, the weight of its outgoing edge is not altered. Note that v_z and all the edges into it are not part of the actual network.
(c) A given vertex v_i, which is not stressed in the current iteration of **L**, could get stressed during future iterations.
(d) In a single iteration of **L**, each vertex is touched exactly once and each edge is touched at most three times (once as an *in-edge*, once as an *out-edge* and once during cost modification). Thus each iteration can be implemented in $O(m)$ time, from which it follows that the Stressing Algorithm runs in $O(m \cdot n)$ time.
(e) The Stressing Algorithm does not alter the topology of the network **G**; however, for all intents and purposes, the cost structure of the input network is irretrievably destroyed. We use $\mathbf{G}_i$ to denote the network after the execution of $\mathbf{L}_i$. When the state of the network is not paramount, we use **G** to denote the network.

4.1. Space requirements

Observe that there are no storage variables in either Algorithm 4.1 or Algorithm 4.2. In other words, the Stressing Algorithm takes zero extra space. This is in marked contrast to the label-correcting algorithms, described in [3], which use $\Omega(n)$ extra space and the contraction-based algorithm, described in [18], which could take $\Omega(n^2)$ extra space, in the worst case. Technically, we need two registers: (a) to track the **for** loop index, and (b) for finding the edge with the least cost. However, any reasonable architecture should provide this space, obviating the need for extra RAM space.

The zero extra-space feature of the Stressing Algorithm finds applications in a number of domains such as scheduling web requests and checking consistency in wireless protocols [14].

5. Preliminaries

In this section, we define some of the terms and state some theorems from polyhedral combinatorics that we shall be using in the rest of the paper.

Definition 5.1. A constraint of the form $x_i - x_j \leqslant c$ is called a difference constraint.

Definition 5.2. A conjunction of difference constraints is called a system of difference constraints or a difference constraint system (DCS).

Let $\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}$ denote a DCS on n variables and m constraints, i.e., **A** has n columns and m rows. It is known that **A** does not have full column rank and that $\text{rank}(\mathbf{A}) \leqslant n - 1$. To see this, observe that adding the columns together results in the $\vec{\mathbf{0}}$ vector, which indicates that the columns are linearly dependent and hence the rank of **A** is at most $n - 1$.

Definition 5.3. A matrix **A** is said to be Totally Unimodular (TUM), if every single square submatrix in **A** has determinant 0, 1 or -1.

It goes without saying that each entry in $\mathbf{A}$ must belong to the set $\{0, 1, -1\}$.

Theorem 5.1. *If $\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}$ is a DCS, then $\mathbf{A}$ is TUM.*

Proof. Let $\mathbf{A}$ be an $m \times n$ matrix, with each row having precisely one $+1$ and precisely one -1.

Choose any $k \times k$ square submatrix in $\mathbf{A}$ and call it $\mathbf{B}$. If every row in $\mathbf{B}$ has both a $+1$ and a -1, then as per the observation above, $\mathbf{B}$ cannot have full rank and hence $\det(\mathbf{B}) = 0$.

So there must be a row in $\mathbf{B}$ with only one non-zero entry; without loss of generality, assume that it is $+1$. With only a loss of sign of $\det(\mathbf{B})$, we can permute the rows and columns of $\mathbf{B}$ so that then non-zero entry is $B[1, 1]$, with all the other entries in row 1 being 0. It is now easy to see that $\det(\mathbf{B}) = \pm \det(\mathbf{B}')$, where $\mathbf{B}'$ is the $(k-1) \times (k-1)$ square submatrix obtained from $\mathbf{B}$ by deleting its first row and first column. Using induction, it is now straightforward to see that $\det(\mathbf{B})$ is 0, 1 or -1. □

Definition 5.4. Give a set S and a binary operation r that is defined on the elements of S, S is said to be closed under r, if the result of applying r to two arbitrary elements x and y in S, results in an element z, which is also in S.

6. Correctness

We now establish the correctness of Algorithm 4.1; the arguments used in the proof, require a clear understanding of the connections between networks and systems of difference constraints.

Lemma 6.1. *Let $r_1 : \{\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}\}$ denote a system of difference constraints and let S' denote the set of solutions to this system. Likewise, let $r_2 : \{\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}},\ \vec{\mathbf{x}} \leqslant \vec{\mathbf{0}}\}$ denote another constraint system and let S denote the set of its solutions. Then, $S' \neq \emptyset$ if and only if $S \neq \emptyset$.*

Proof. Since the constraints in r_2 are a superset of the constraints in r_1, any solution in S is also a solution in S' and thus $S \neq \emptyset \rightarrow S' \neq \emptyset$. Let us now assume that $S' \neq \emptyset$ and let $\vec{\mathbf{x}}$ denote an arbitrary element of S'. Since r_1 is defined purely by difference constraints, it must be the case that $\vec{\mathbf{x}} - \vec{\mathbf{d}}$ is also a member of S', where d is a positive integer and $\vec{\mathbf{d}} = [d, d, \ldots, d]^T$. We can always choose d sufficiently large to ensure that all components of $\vec{\mathbf{x}} - \vec{\mathbf{d}}$ are non-positive. It follows that $S \neq \emptyset$ and thus we have $S' \neq \emptyset$ if and only if $S \neq \emptyset$. □

Algorithm 6.1 is an incremental approach for determining the feasible solution of the constraint system $r_2 : \{\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}},\ \vec{\mathbf{x}} \leqslant \vec{\mathbf{0}}\}$, where $\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}$ is a difference constraint system; we use S to denote the set of solutions of r_2.

Observe that, as specified, Algorithm 6.1 is a non-terminating procedure, since if the solution set S to the initial constraint system is empty, then it will recurse forever. We proceed to show that if $S \neq \emptyset$, then Algorithm 6.1 definitely terminates. $\vec{\mathbf{o}}$ represents the origin of the current affine space; when the algorithm is called for the first time, $\vec{\mathbf{o}}$ is initialized to $\vec{\mathbf{0}}$.

Function INCREM-DIFF ($\mathbf{A}$, $\vec{\mathbf{c}}$, $\vec{\mathbf{o}}$)

1: {Note that the constraint system that we are trying to solve is $\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}},\ \vec{\mathbf{x}} \leqslant \vec{\mathbf{0}}$.
Further, the origin of the affine space is $\vec{\mathbf{o}} = [o_1, o_2, \ldots, o_n]^T$ }. Initially, $\vec{\mathbf{o}} = \vec{\mathbf{0}}$.
2: **if** ($\vec{\mathbf{c}} \geqslant \vec{\mathbf{0}}$) **then**
3: Set $\vec{\mathbf{x}} = \vec{\mathbf{o}}$.
4: **return**($\vec{\mathbf{x}}$)
5: **end if**
6: Find a constraint l' with a negative Right-Hand Side (RHS).
7: Let $l' : x_i - x_j \leqslant c_{ij},\ c_{ij} < 0$, denote this constraint.
8: Replace the variable x_i by the variable $x_i' = x_i - c_{ij}$, in each constraint where x_i occurs.
9: Set $o_i = o_i + c_{ij}$.
10: Let $\mathbf{A}' \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}'$ denote the new constraint system.
11: INCREM-DIFF($\mathbf{A}'$, $\vec{\mathbf{c}}'$, $\vec{\mathbf{o}}$).

Algorithm 6.1. The Incremental Algorithm for a System of Difference Constraints.

Definition 6.1. Given a non-empty set of vectors $\mathcal{S}$, a vector $\vec{\mathbf{y}} \in \mathcal{S}$, is said to be a maximal element, if $\vec{\mathbf{x}} \in S \Rightarrow \vec{\mathbf{y}} \geqslant \vec{\mathbf{x}}$, where the "$\geqslant$" relation holds componentwise.

It is not hard to see that if a set contains a maximal element, then this element is unique. Two elements $\vec{\mathbf{u}}$ and $\vec{\mathbf{v}}$ in S are incomparable, if neither $\vec{\mathbf{u}} \geqslant \vec{\mathbf{v}}$ nor $\vec{\mathbf{v}} \geqslant \vec{\mathbf{u}}$.

We remark that our definition of maximal element is different from the standard definition of maximal element; in the standard definition, an element of a set $\vec{\mathbf{y}}$ is declared to be maximal, as long as there is no element $\vec{\mathbf{z}}$, such that $\vec{\mathbf{z}} \geqslant \vec{\mathbf{y}}$ and $z_i > y_i$, for at least one $i = 1, 2, \ldots, n$. In other words, as per the standard definition, a set could have multiple maximal elements, which are mutually incomparable. We will be using our definition for the rest of the paper.

Lemma 6.2. *Given a non-empty set S of vectors, which is closed and bounded above, and a partial order "$\geqslant$" defined on the elements of S, either S has a maximal element $\vec{\mathbf{z}}$, or there exists a pair of elements $\vec{\mathbf{u}}$, $\vec{\mathbf{v}} \in S$, such that $\vec{\mathbf{u}}$ and $\vec{\mathbf{v}}$ are incomparable and there is no element $\vec{\mathbf{z}} \in S$, such that $\vec{\mathbf{z}} \geqslant \vec{\mathbf{u}}$ and $\vec{\mathbf{z}} \geqslant \vec{\mathbf{v}}$.*

Proof. Suppose that S does not have a maximal element. Then, as per our definition, there must be at least two elements, say $\vec{\mathbf{u}}$ and $\vec{\mathbf{v}}$ which are incomparable, since if every pair of elements is comparable, then the elements would form a chain, under the "$\geqslant$" relationship and every chain which is bounded above, has a maximal element. Now, consider the case in which corresponding to every pair of incomparable elements (say $(\vec{\mathbf{u}}, \vec{\mathbf{v}})$), there is an element $\vec{\mathbf{z}} \in S$, such that $\vec{\mathbf{z}} \geqslant \vec{\mathbf{u}}$ and $\vec{\mathbf{z}} \geqslant \vec{\mathbf{v}}$. We call $\vec{\mathbf{z}}$ the dominator of $\vec{\mathbf{u}}$ and $\vec{\mathbf{v}}$. Observe that we can create a set of dominators of all pairs of elements that are mutually incomparable; either the elements of this set form a chain or we can create the set of their dominators. As this process repeats, we will be left with a single element, since S is closed and bounded above. This single element is clearly the maximal element of S, violating the assumption that S did not have a maximal element. □

Remark 6.1. For integral $\vec{\mathbf{c}}$, the solution set S of the difference constraint system $\{\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}},\ \vec{\mathbf{x}} \leqslant \vec{\mathbf{0}}\}$ is non-empty, if and only if it contains lattice points, as per the consequences of total unimodularity [15]. The set of lattice points in S is a discrete, closed set, which is bounded above by $\vec{\mathbf{0}}$. From this point onwards, we shall focus on this set i.e., the set of lattice point solutions only, when the non-emptiness of S is discussed.

Lemma 6.3. *The solution set S of the constraint system $\{\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}},\ \vec{\mathbf{x}} \leqslant \vec{\mathbf{0}}\}$, where $(\mathbf{A},\ \vec{\mathbf{c}})$ is a system of difference constraints, contains a maximal element, if $S \neq \emptyset$.*

Proof. If S contains a unique element, the lemma is trivially true. Assume that S contains more than one element and that it does not have a maximal element. We observe that the set S is bounded above by $\vec{\mathbf{0}}$.

Clearly, if every pair of elements in S is comparable, then these elements form a chain under the componentwise "$\geqslant$" relationship and there must exist a maximal element in S. Since, S does not have a maximal element, as per Lemma 6.2, it contains at least two elements, say, $\vec{\mathbf{u}}$ and $\vec{\mathbf{v}}$, which are incomparable and further there is no element $\vec{\mathbf{z}} \in S$, such that $\vec{\mathbf{z}} \geqslant \vec{\mathbf{u}},\ \vec{\mathbf{v}}$. We shall now demonstrate that such a vector must exist in S, contradicting the consequences of the hypothesis that S does not have a maximal element.

Construct the vector $\vec{\mathbf{z}}$ by taking the componentwise maximum of $\vec{\mathbf{u}}$ and $\vec{\mathbf{v}}$; i.e., $z_i = \max(u_i, v_i)$. We shall show that $\vec{\mathbf{z}} \in S$.

Let $l_1 : x_i - x_j \leqslant c_{ij}$ denote an arbitrary constraint defining S. Since $\vec{\mathbf{u}}$ and $\vec{\mathbf{v}}$ are in S, we must have:

$$\begin{aligned} u_i - u_j &\leqslant c_{ij} \\ v_i - v_j &\leqslant c_{ij} \end{aligned} \tag{1}$$

Without loss of generality, assume that $u_j \geqslant v_j$; thus $u_j = \max(u_j, v_j)$. Since $u_i - u_j \leqslant c_{ij}$, it follows that $\max(u_i, v_i) - u_j \leqslant c_{ij}$, and hence, $\max(u_i, v_i) - \max(u_j, v_j) \leqslant c_{ij}$.

The constraint l_1 was chosen arbitrarily; we can therefore apply our analysis to every constraint. In other words setting $z_i = \max(u_i, v_i),\ \forall i = 1, 2, \ldots, n$ gives a solution to the constraint system, i.e., $\vec{\mathbf{z}} \in S$. It follows that the lattice of the elements of S under the componentwise "$\geqslant$" relationship contains a maximal element, as per our definition of maximal element. □

We need the following result on Total Unimodularity, which we state without proof.

Result 6.1. *Let* $\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{b}}$ *denote a linear system with* $\vec{\mathbf{b}}$ *integral and* $\mathbf{A}$ *Totally Unimodular. Then* $\max_{\mathbf{A}\cdot\vec{\mathbf{x}}\leqslant\vec{\mathbf{b}}} \vec{\mathbf{c}} \cdot \vec{\mathbf{x}}$ *has an integral solution.*

Theorem 6.1. *If the solution set* S *of the constraint system* $\{\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}, \vec{\mathbf{x}} \leqslant \vec{\mathbf{0}}\}$ *is non-empty, then Algorithm 6.1 terminates by returning the maximal element of* S.

Proof. Observe that if $S \neq \emptyset$, then as per Lemma 6.3, it contains a unique maximal element, say $\vec{\mathbf{u}} \leqslant \vec{\mathbf{0}}$. Since $\vec{\mathbf{c}}$ is an integral vector, we are guaranteed that $\vec{\mathbf{u}}$ is integral, by Result 6.1.

Consider the case, in which no recursive calls are made and line 3 of Algorithm 6.1 is executed on the initial invocation; in this case, $\vec{\mathbf{o}} = \vec{\mathbf{0}}$. Note that if $\vec{\mathbf{c}} \geqslant \vec{\mathbf{0}}$, then $\vec{\mathbf{0}}$ is clearly a solution to the constraint system and hence belongs to S. Additionally, $\vec{\mathbf{0}}$ is the unique maximal element of S, i.e., $\vec{\mathbf{u}} = \vec{\mathbf{o}} = \vec{\mathbf{0}}$, and hence, the theorem holds.

We now consider the case in which one or more recursive calls are made within Algorithm 6.1.

Let $S_0 = \{\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}, \vec{\mathbf{x}} \leqslant \vec{\mathbf{0}}\}$ denote the constraint system when INCREM-DIFF() is called for the first time. Since, $\vec{\mathbf{c}} \not\geqslant \vec{\mathbf{0}}$, we have a constraint of the form: $x_i - x_j \leqslant c_{ij},\ c_{ij} < 0$. Without loss of generality, we assume that x_i is replaced by the variable $x_i'(= x_i - c_{ij})$, in all the constraints of S_0, to get the new constraint system $S_1 = \{\mathbf{A}' \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}', \vec{\mathbf{x}} \leqslant \vec{\mathbf{0}}\}$. Since $x_j \leqslant 0$, the constraint $x_i - x_j \leqslant c_{ij}$ clearly implies that $x_i \leqslant c_{ij}$, *in any solution to the constraint system*. Accordingly, replacing the variable x_i by the variable $x_i' = x_i - c_{ij}$, where $x_i' \leqslant 0$, does not alter the solution space. In other words, S_0 is feasible, if and only if S_1 is. The polyhedron defining S_0 in the initial affine space, has been shifted to a new affine space, in which the ith component of the origin has moved from o_i to $o_i + c_{ij}$. From the mechanics of the translation, it is clear that there is a one-to-one correspondence, between the elements of S_0 and S_1, which preserves the componentwise "$\geqslant$" relationship. It follows that the maximal element of S_0 is translated to the maximal element of S_1. Hence, Algorithm 6.1 maintains the following invariant: $\vec{\mathbf{o}} \geqslant \vec{\mathbf{y}},\ \forall \vec{\mathbf{y}} \in S$.

During each recursive call made by Algorithm 6.1, some component of the origin $\vec{\mathbf{o}}$ is decreased by at least unity and hence after at most $n \cdot \|u\|_\infty$ recursive calls, we must have $\vec{\mathbf{o}} \leqslant \vec{\mathbf{u}}$, where $\|u\|_\infty$ denotes the largest absolute value over all components of $\vec{\mathbf{u}}$. But, by construction, $\vec{\mathbf{o}} \geqslant \vec{\mathbf{y}}$, for all $\vec{\mathbf{y}} \in S$, and therefore, we must have $\vec{\mathbf{o}} = \vec{\mathbf{u}}$. □

Observe that if a polyhedron $\mathbf{P}$ has a unique maximal element, then this element is obtained by maximizing the linear function $\vec{\mathbf{p}} \cdot \vec{\mathbf{x}}$ over $\mathbf{P}$, where $\vec{\mathbf{p}} > \vec{\mathbf{0}}$ is an arbitrary positive vector. Therefore, without loss of generality, we can assume that Algorithm 6.1 is in essence, solving the following linear program:

$$\begin{aligned} \Psi: \quad & \max \sum_{i=1}^{n} x_i \\ & \mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}} \\ & \vec{\mathbf{x}} \leqslant \vec{\mathbf{0}} \end{aligned} \tag{2}$$

Given a network $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \vec{\mathbf{c}} \rangle$, it is a straightforward task to construct the constraint system $\{\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}\}$, with solution set S', such that $S' = \emptyset$, if and only if $\mathbf{G}$ contains a negative cost cycle (see chapter on Single-source shortest paths in [4]). The construction consists of two steps:

(a) Corresponding to each vertex v_i, we create a variable x_i;
(b) Corresponding to each arc $v_j \rightsquigarrow v_i$ with cost c_{ij}, we create the constraint $x_i - x_j \leqslant c_{ij}$.

It is clear that if $\mathbf{G}$ has m arcs and n nodes, then $\mathbf{A}$ will have m rows and n columns.

From Lemma 6.1 and the above observation, we know that $\mathbf{G}$ contains a negative cost cycle, if and only if, the solution set S to the constraint system $\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}} \wedge \vec{\mathbf{x}} \leqslant \vec{\mathbf{0}}$ is empty. From this point onwards, we shall refer to System (2) as the constraint system corresponding to the network $\mathbf{G}$.

We next observe that Algorithm 4.1 as applied to the network $\mathbf{G}$, is precisely the application of Algorithm 6.1 to the corresponding system of difference constraints, viz., System (2). The STRESS() operation applied to a vertex is equivalent to replacing the variable x_i with the variable $x_i - c_{ij}$, in every constraint in which x_i occurs. Therefore, the

phrases "If **G** does not contain a negative cost cycle" and "If System (2) is feasible", denote the same state of events. The key differences between the two algorithms are as follows:

(i) Algorithm 4.1 is a actually a terminating procedure, which concludes in exactly $O(m \cdot n)$ steps, whether or not the constraint system corresponding to the input network is feasible, whereas Algorithm 6.1 will not terminate, if the input system is infeasible.
(ii) In Algorithm 4.1, we do not store the value of the origin, as is done in Algorithm 6.1. However, in order to simplify the exposition of the proof of correctness of Algorithm 4.1, it is helpful to associate the value o_i of the constraint System (2), with vertex v_i of the network **G**.
(iii) In Algorithm 6.1, the variable to be replaced is chosen arbitrarily. In Algorithm 4.1, all variables are stressed in each round, if they can be stressed. Further, to stress a vertex, we select an incoming edge of least cost, i.e., there is a greediness to our approach.

Our proof of correctness will establish that stressing every vertex, in each of the $(n-1)$ rounds, is sufficient to detect infeasibility in System (2), in the following sense: If there exists a negative cost edge in the network **G**, after line 9 of Algorithm 4.1 has been executed for the final time, then System (2) is infeasible. However, this immediately establishes that the network **G** contains a negative cost cycle.

We reiterate that a vertex is stressed by Algorithm 4.1, only if it has an incoming edge with negative cost.

Lemma 6.4. *If System (2) is feasible, then $o_i = 0$ for some vertex v_i.*

Proof. Let $\vec{\mathbf{z}}$ denote the solution to System (2), and let $z_i < 0,\ \forall i$. We use Ψ_z to denote the value of the objective function $\sum_{i=1}^{n} x_i$ at this point. Let $k = \max_{i=1}^{n} z_i$. Observe that $\vec{\mathbf{u}} = (\vec{\mathbf{z}} - \vec{\mathbf{k}})$ is also a solution to System (2), and $\vec{\mathbf{u}} > \vec{\mathbf{z}}$. Hence $\Psi_u > \Psi_z$, thereby contradicting the optimality of $\vec{\mathbf{z}}$. The lemma follows. □

Claim 6.1. *For each $i = 1, 2, \ldots, n$, the o_i value in Algorithm 6.1 decreases monotonically with each recursive call.*

Proof. Observe that the only operation performed on the o_i values, is the addition of a negative number on line 8 of Algorithm 6.1. □

We use S_i to denote the set of vertices that are stressed during $\mathbf{L}_i$ of Algorithm 4.1. By convention, $S_0 = \mathbf{V}$, i.e., we say that all the vertices of the network are stressed during $\mathbf{L}_0$. Thus, if a vertex is stressed only during $\mathbf{L}_0$, it means that the vertex is not stressed at all.

Definition 6.2. A vertex in the network **G** is said to be saturated at level i, if it is stressed during $\mathbf{L}_i$, but never afterwards during Algorithm 4.1.

We use Z_i to denote the set of vertices which are saturated at level i. It is important to note that there could exist a vertex $v_a \in \mathbf{G}$, such that $v_a \in S_i$, but $v_a \notin Z_i$. In other words, a vertex which is stressed during $\mathbf{L}_i$ need not necessarily be saturated at level i. However, if $v_a \in Z_i$, then v_a is necessarily part of S_i, since v_a is stressed during $\mathbf{L}_i$. We thus have $Z_i \subseteq S_i,\ \forall i = 0, 1, \ldots, (n-1)$.

Lemma 6.5. *If System (2) is feasible, then there exists at least one vertex v_i, which is never stressed by Algorithm 4.1.*

Proof. Observe that when vertex v_i is stressed, o_i drops in value. From Claim 6.1, we know that o_i can never increase for any vertex v_i. Thus, if System (2) is feasible and all the vertices of **G** are stressed at least once, by Algorithm 4.1, then on its termination, $o_i < 0,\ \forall i$, thereby contradicting Lemma 6.4. □

Lemma 6.5 establishes that there is at least one vertex which is saturated at level 0.

Lemma 6.6. *If $S_i = \emptyset$, then $S_j = \emptyset,\ j = (i+1), (i+2), \ldots, (n-1)$. Further, System (2) is feasible and* **G** *does not have a negative cost cycle.*

Proof. If no vertex was stressed during $\mathbf{L}_i$, then no vertex had an incoming edge of negative cost, in the current network and hence the current network does not have a negative edge at all. This situation will not change at the commencement of $\mathbf{L}_{i+1}$ and hence S_j will be empty as well, for $j = (i+1), \ldots, (n-1)$. Let $\mathbf{T} : \mathbf{A}' \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}'$ denote the constraint system corresponding to the network $\mathbf{G}_i$, i.e., the network that results after $\mathbf{L}_i$ completes execution. Since $\mathbf{G}_i$ does not have a negative cost edge, $\vec{\mathbf{c}}' \geqslant \vec{\mathbf{0}}$. Therefore, $\mathbf{T}$ is feasible. However, as noted in Theorem 6.1, if $\mathbf{T}$ is feasible, then so is System (2), which is the constraint system corresponding to the network $\mathbf{G}_0$. This immediately implies that $\mathbf{G}_0$ and hence $\mathbf{G}$, do not have negative cost cycles. □

Theorem 6.2. *Assume that* $\mathbf{G}$ *does not have a negative cost cycle. Then,*

$$(S_i \neq \emptyset) \rightarrow (Z_i \neq \emptyset), \quad i = 0, 1, 2, \ldots n.$$

Proof. By convention, $S_0 = \mathbf{V}$ and by Lemma 6.5, we know that $Z_0 \neq \emptyset$; so the theorem is true for $i = 0$.

In order to extend the theorem for higher values of i, we need to understand the structure of Z_0. Let us focus on the constraint system corresponding to $\mathbf{G}$, i.e., System (2). We know that the objective function, Ψ, is maximized at a minimal face of the polyhedron $\{\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}, \vec{\mathbf{x}} \leqslant \vec{\mathbf{0}}\}$, say F. From [15, page 101], we know that $F = \{\vec{\mathbf{x}} \colon \mathbf{B} \cdot \vec{\mathbf{x}} = \vec{\mathbf{c}}'\}$, where $\{\mathbf{B} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}'\}$ is a subsystem of the system $\{\mathbf{A} \cdot \vec{\mathbf{x}} \leqslant \vec{\mathbf{c}}, \vec{\mathbf{x}} \leqslant \vec{\mathbf{0}}\}$. We have already established that System (2) has a unique maximal point, i.e., the minimal face, F, corresponding to Ψ, is actually a vertex. This implies that the matrix $\mathbf{B}$ in the constraint system describing F is a basis, i.e., $\text{rank}(\mathbf{B}) = n$. We shall refer to $\mathbf{B}$ as the optimal basis of System (2).

We reiterate that F is defined by a collection of difference constraint *equalities* along with one or more constraints of the form $x_i = 0$, which we term absolute constraints. Note that all the constraints in the optimal basis cannot be difference constraints, since such a matrix has rank at most $(n-1)$ and $\mathbf{B}$ is a basis. In the network $\mathbf{G}$, the arcs corresponding to the difference constraints in $\mathbf{B}$, form a tree, $\mathbf{B}_t$.

We associate a vertex set M_i, $i = 0, 1, \ldots, (n-1)$, with loop $\mathbf{L}_i$; these sets will be populated inductively.

We define the set M_0 as follows:

(i) Vertex $v_j \in M_0$, if the constraint $x_j = 0$ is one of absolute constraints of the basis $\mathbf{B}$.
(ii) If vertex $v_j \in M_0$ and the constraint $x_k - x_j = 0$ is one of the constraints of the basis $\mathbf{B}$, then vertex $v_k \in M_0$.

We shall now establish that $M_0 = Z_0$, i.e., M_0 is precisely the set of vertices that are stressed only during $\mathbf{L}_0$ by Algorithm 4.1.

We focus on $\mathbf{L}_0$. Let v_j denote an arbitrary vertex in M_0, such that $x_j = 0$ is an absolute constraint of $\mathbf{B}$. Note that v_j cannot have an incoming edge of negative cost. To see this, we assume the contrary and arrive at a contradiction. Let us say that there is a negative cost edge into v_j, having cost b, $b < 0$. From the correctness of Algorithm 6.1, we know that $o_j \leqslant b$ and hence $x_j \leqslant b$, in any solution of the System (2). But this contradicts the feasibility of the basis $\mathbf{B}$, from which it follows that v_j cannot have an incoming edge of negative cost. We now consider a vertex $v_k \in M_0$, such that $x_k - x_j = 0$ is a constraint of the basis $\mathbf{B}$ and $v_j \in M_0$. Using an identical argument, it is clear that v_k does not have an incoming edge of negative cost either, since x_k must be 0, in the optimal solution. We have thus established that none of the vertices in M_0 have incoming negative edges, at the commencement of Algorithm 4.1. Let v_j be an arbitrary vertex in M_0. Let us say that during some iteration of $\mathbf{L}$, the edge cost of an edge coming into v_j from some vertex v_k becomes negative (say $b < 0$). This would imply that $x_j - x_k \leqslant b$ is a constraint derived by Algorithm 6.1 and hence $x_j \leqslant b$ must hold in any feasible solution of System (2), contradicting the feasibility of $\mathbf{B}$. In other words, the vertices in M_0 are *never* stressed by Algorithm 4.1. It is thus clear that $M_0 = Z_0$.

The set M_1 is defined as follows:

(i) Vertex $v_j \in M_1$, if the constraint $x_j - x_a = b$, $b \neq 0$ is a constraint of the basis $\mathbf{B}$, where $v_a \in M_0$.
(ii) If vertex $v_j \in M_1$ and the constraint $x_k - x_j = 0$ is a constraint of the basis $\mathbf{B}$, then $v_k \in M_1$.

We shall show that $M_1 = Z_1$, i.e., M_1 is precisely the set of vertices that are saturated at level 1.

Consider the case in which $M_1 = \emptyset$. This means that there are no constraints in the basis $\mathbf{B}$, having the form $x_k - x_a = c_{ka}$, where $x_a \in M_0$. Delete all the constraints (rows and columns) corresponding to the vertices in M_0

from $\mathbf{B}$ to get the constraint system $\{\mathbf{B}_1 \cdot \vec{\mathbf{x}}_1 = \vec{\mathbf{b}}_1\}$. Observe that the deletion of these constraints preserves the basis structure; on the other hand, $\mathbf{B}_1$ is constituted exclusively of difference constraints and hence cannot be a basis. The only conclusion that can be drawn is that there are no constraints in $\{\mathbf{B}_1 \cdot \vec{\mathbf{x}}_1 = \vec{\mathbf{b}}_1\}$, i.e., $Z_0 = \mathbf{V}$. Thus, $S_1 = \emptyset$ and the theorem is trivially true.

We now handle the case in which $M_1 \neq \emptyset$. Focus on $\mathbf{L}_1$ and consider a constraint of the form $x_j - x_a = c_{ja}$ in the basis $\mathbf{B}$, where $v_a \in M_0$ and $c_{ja} \neq 0$. Since $\mathbf{B}$ is a feasible basis and $x_a = 0$, we must have $c_{ja} < 0$ and $x_j = c_{ja}$. As per the construction of the constraint network, there exists an edge $v_a \rightsquigarrow v_j$ with cost c_{ja}. We now claim that no edge into v_j can have cost lower than c_{ja}. To see this, observe that the cost of edge $v_a \rightsquigarrow v_j$ is altered only through stressing v_j, since v_a is never stressed in the algorithm. Assume that there exists an edge into v_j, with cost strictly less than c_{ja}, say c'_{ja}. During $\mathbf{L}_1$, v_j will be stressed; but this means that $o_j \leqslant c'_{ja} < c_{ja}$, in any feasible basis, thereby contradicting the feasibility of the basis $\mathbf{B}$. When v_j is stressed during $\mathbf{L}_1$, o_j reaches its correct value, viz., c_{ja}, because we stress v_j using the least cost edge into it. o_j stays at this value over all future iterations, since any additional STRESS() operation on v_j will decrease o_j, contradicting the feasibility of $\mathbf{B}$. In other words, v_j is never stressed again and thus saturated at level 1. Using an identical argument, we can establish that a vertex $v_j \in M_1$, such that $x_j - x_a = 0$ is a constraint of $\mathbf{B}$ and $v_a \in M_1$ will also be saturated at level 1. It follows that $M_1 = Z_1 \neq \emptyset$ and the theorem follows.

Now, observe that once the o values of the vertices in M_1 are determined, the constraints in $\mathbf{B}$ having the form $x_r - x_j = c_{rj}$, where $x_j \in M_1$ become absolute constraints, since x_j has been fixed in the current iteration.

This argument can be applied inductively for $i = 2, 3, \ldots, (n-1)$ as follows:

(i) Let M_i denote the set of vertices v_j, such that either there is a constraint $x_j - x_a = c_{ja}$ in $\mathbf{B}$, where $v_a \in M_{i-1}$, or there is a constraint $x_j - x_a = 0$, where v_a is already in M_i.

(ii) If $M_i = \emptyset$. the elimination of the constraints, in $\bigcup_{j=1}^{i-1} M_j$ from $\mathbf{B}$, to get $\mathbf{B}_i$, should preserve the basis structure of $\mathbf{B}$; however, a system of pure difference constraints, such as $\mathbf{B}_i$, cannot form a basis and therefore, the resultant constraint system should be empty, i.e., $\bigcup_{j=1}^{(i-1)} M_j = \mathbf{V}$. This implies that $Z_i = S_i = \emptyset$ and the theorem is true.

(iii) If $M_i \neq \emptyset$, then $\mathbf{L}_i$ fixes the o values of the vertices in M_i to their final values, so that these vertices are never stressed again. In other words, $M_i = Z_i \neq \emptyset$ and hence the theorem holds. □

We recall that $\mathbf{G}_i$ is the network that results after the ith round of stressing.

Theorem 6.3. *If $\mathbf{G}_i$ has a negative cost cycle, then so does $\mathbf{G}_{i+1}$, $i = 0, 1, \ldots, (n-1)$.*

Proof. Recall that Algorithm 4.1 alters only the cost structure of the input network $\mathbf{G}$ and not its topology. Accordingly, each cycle in $\mathbf{G}_i$ is also a cycle in $\mathbf{G}_{i+1}$ and vice versa. The key observation is that the STRESS() operation, applied to a vertex, preserves the cost of all cycles in $\mathbf{G}_i$.

Let R denote an arbitrary cycle in $\mathbf{G}_i$, with cost $c(R)$. Consider the application of a STRESS() operation, to a vertex v_i in $\mathbf{G}_i$. Clearly, the cost of the cycle R is not affected, if v_i is not on R.

Fig. 1 demonstrates the case, in which $v_i \in R$. Focus on the two edges e_1 and e_2, which enter and leave v_i on R respectively.

As per the mechanics of the STRESS() operation, a negative number is subtracted from the cost of e_1 and the same number is added to the cost of e_2. It follows that the sum of the costs of the edges around the cycle R remains the same as before, i.e., $c(R)$ is unaltered.

Since the above argument can be applied for each application of the STRESS() operation, the theorem follows. □

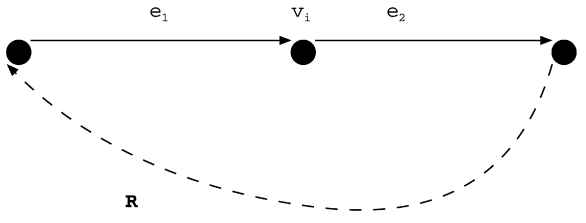

Fig. 1. Stressing a vertex preserves cycle costs.

Theorem 6.4. *The Stressing Algorithm executes line* 11*, if and only if the input network* **G** *contains a negative cost cycle.*

Proof. If the input network, **G**, contains a negative cost cycle, then the STRESS() operations executed in lines (1:)–(9:) of Algorithm 4.1 are not going to change its cost, as per Theorem 6.3. Indeed, after line 9 is executed for the last time, **G** will continue to have a negative cost cycle and hence a negative cost edge. This negative cost edge will be detected in line 10 of the algorithm and hence line 11 will be executed.

On the other hand, if **G** does not contain a negative cost cycle, then as argued previously, the constraint system, defined by System (2) is feasible and has a unique optimal solution. We need to consider the following two cases:

(i) $S_i = \emptyset$, for some $i \in \{1, 2, \ldots, (n-1)\}$.
(ii) $S_i \neq \emptyset$, $\forall i \in \{1, 2, \ldots, (n-1)\}$.

In the former case, there were no vertices to be stressed during $\mathbf{L}_i$, for some $i = 0, 1, \ldots, (n-1)$. This means that all the incoming edges of all the vertices are non-negative. However, this is only possible, if all the edges in the network are non-negative. This implies that line 11 will not be executed.

In the latter case, $|Z_i| \geqslant 1$, $i = 0, 1, \ldots, (n-1)$ and hence $\bigcup_{i=0}^{n-1} Z_i = \mathbf{V}$. This means that all n vertices of the network have been saturated at some level, between 0 and $(n-1)$, i.e., the o value corresponding to each vertex has reached its final value. Hence, no vertex can be stressed any longer and all the incoming edges of each vertex are non-negative. This implies that line 11 will not be executed. □

We observe that in an adhoc networks setting, a node on a negative cost cycle has to merely monitor the cost of its incoming edges to determine that the given set of constraints is infeasible.

7. Computing shortest paths

In this section, we argue that in the absence of a negative cost cycle, Algorithm 4.1, as specified, does not provide a solution to the Single-Source Shortest paths problem.

For instance, consider the network in Fig. 2.

Observe that only v_3, v_4 and v_5 have incoming edges and the edge into v_5 has positive cost. Stressing v_3 gives us the network in Fig. 3.

Stressing v_4 gives the network in Fig. 4.

Finally stressing v_5 gives the network in Fig. 5.

It is not hard to check that $\vec{\mathbf{o}} = [0, 0, -3, -7, -1]^T$ at the termination of Algorithm 4.1. These values do not represent the Single Source Shortest Path distances from either v_1 or v_2. What is more significant is that it is completely unclear, how the Stressing Algorithm will orient the network towards determining shortest paths from a specified source, since as specified, the algorithm blindly stresses every vertex with an incoming negative edge, in each iteration.

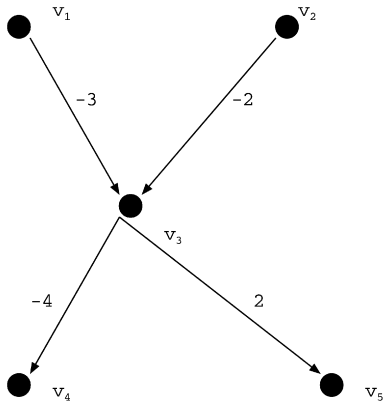

Fig. 2. Stressing and shortest paths.

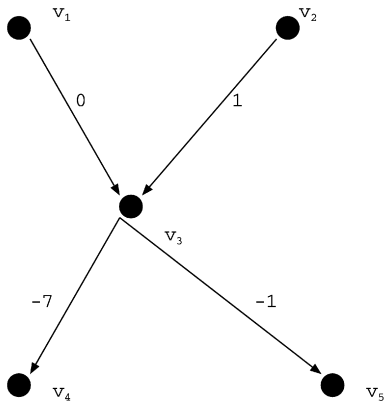

Fig. 3. Network after stressing v_3.

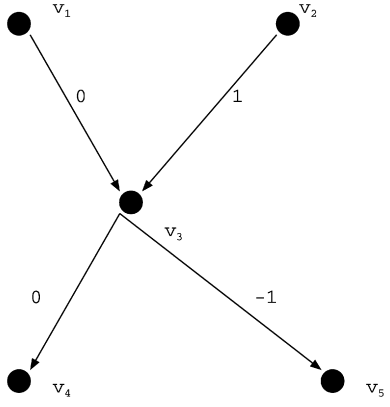

Fig. 4. Network after stressing v_4.

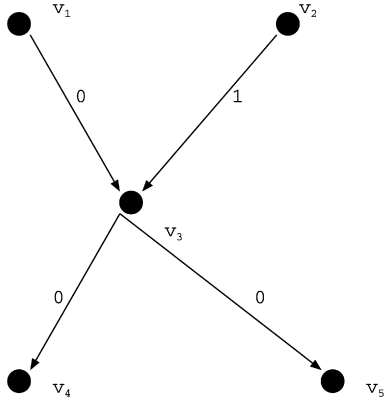

Fig. 5. Network of Fig. 2 after one complete iteration.

8. Contrasting the Stressing Algorithm with Relaxation-based approaches

The key features of the Stressing Algorithm are as follows:

(a) The Stressing Algorithm is a greedy approach, whereas Relaxation-based approaches are based on Dynamic Programming. Observe that the proof of correctness of the Stressing Algorithm depends on the fact that when a vertex is stressed, the incoming edge with the least cost is selected.

(b) The Stressing Algorithm is "sourceless" and hence can be easily distributed; the Relaxation-based approaches are Single-Source Shortest Path algorithms. As argued in the previous section, even if distance labels were permitted, the Stressing algorithm does not produce Shortest Paths from any vertex. *This is an especially important feature in application areas such as Adhoc Networks*.
(c) The Stressing Algorithm does not require any labeling space; on the other hand, it is not immediately clear, as to how the negative cost cycle can be extracted.

9. Conclusions

In this paper, we introduced a new technique for discovering the existence of negative cost cycles in networks with positive and negative arc costs. The novelty of our approach is that it takes *zero* extra space; all other algorithms that are cited in the literature use $\Omega(n)$ extra space, in the worst case. It must be noted that the Stressing Algorithm is as efficient as the fastest known algorithm for the NCCD problem. We believe that the arguments used in proving the correctness of the stressing approach will find applications in the design of algorithms for other network optimization problems, such as Min-cost flow.

We also note that any label-correcting algorithm to determine negative cost cycles requires that all vertices in the network be reachable from the specified source, whereas the Stressing Algorithm does not impose any connectivity requirement on the network.

We plan to extend the work in this paper, along the following avenues:

(i) Building an implementation suite of Stressing Algorithms, in a manner similar to the study in [3].
(ii) Designing a zero-space parallel algorithm for NCCD.
(iii) Modifying the Stressing Algorithm so that the negative cost cycle can be produced, using $O(n)$ space.

Acknowledgements

This research was inspired in part by a grant from the Department of Computer Science, Carnegie Mellon University. The initial draft was formulated at the VLSI CAD Laboratory, University of California, San Diego, where the author was supported by a Visiting Professorship.

We are also grateful to Vineet Bafna and R. Ravi for friendly discussions.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms and Applications, Prentice-Hall, 1993.
[2] R. Alur, D.L. Dill, A theory of timed automata, Theoretical Computer Science 126 (2) (25 April 1994) 183–235, Fundamental Study.
[3] B.V. Cherkassky, A.V. Goldberg, T. Radzik, Shortest paths algorithms: Theory and experimental evaluation, Mathematical Programming 73 (1996) 129–174.
[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, second ed., MIT Press/McGraw-Hill Book Company, Boston, MA, 1992.
[5] I.J. Cox, S.B. Rao, Y. Zhong, Ration regions: A technique for image segmentation, in: Proceedings of the International Conference on Pattern Recognition, IEEE, August 1996, pp. 557–564.
[6] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence 49 (1991) 61–95.
[7] J. Fakcharoenphol, S. Rao, Planar graphs, negative weight edges, shortest paths, and near linear time, in: IEEE (Ed.), 42nd IEEE Symposium on Foundations of Computer Science: Proceedings, October 14–17, 2001, Las Vegas, NV, USA, IEEE Computer Society Press, 2001, pp. 232–241.
[8] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman Company, San Francisco, CA, 1979.
[9] A. Goldberg, Shortest path algorithms: Engineering aspects, in: ISAAC: 12th International Symposium on Algorithms and Computation, 2001, pp. 502–513.
[10] A.V. Goldberg, Scaling algorithms for the shortest paths problem, SIAM Journal on Computing 24 (3) (June 1995) 494–504.
[11] C.C. Han, K.J. Lin, Scheduling real-time computations with separation constraints, Information Processing Letters 12 (May 1992) 61–66.
[12] M. Klein, A primal method for minimal cost flows with applications to the assignment and transportation problems, Management Science (14) (1967) 205–220.
[13] P. Morris, N. Muscatella, T. Vidal, Dynamic control of plans with temporal uncertainty, in: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI), 2001.
[14] R. Ravi, Personal communication.

[15] A. Schrijver, Theory of Linear and Integer Programming, John Wiley and Sons, New York, 1987.
[16] K. Subramani, An analysis of zero-clairvoyant scheduling, in: J.-P. Katoen, P. Stevens (Eds.), Proceedings of the 8th International Conference on Tools and Algorithms for the Construction of Systems (TACAS), in: Lecture Notes in Computer Science, vol. 2280, Springer-Verlag, Berlin, April 2002, pp. 98–112.
[17] K. Subramani, L. Kovalchick, Contraction versus relaxation: A comparison of two approaches for the negative cost cycle detection problem, in: P.M.A. Sloot, et al. (Eds.), Proceedings of the 3rd International Conference on Computational Science (ICCS), in: Lecture Notes in Computer Science, vol. 2659, Springer-Verlag, Berlin, June 2003, pp. 377–387.
[18] K. Subramani, L. Kovalchick, A greedy strategy for detecting negative cost cycles in networks, Future Generation Computer Systems 21 (4) (2005) 607–623.
[19] R.E. Tarjan, Data Structures and Network Algorithms, CBMS-NSF Reg. Conf. Ser. Appl. Math., vol. 44, SIAM, 1983.