

# Verification in Optimal Space

K. Subramani,  
LDCSEE,  
West Virginia University,  
Morgantown, WV  
{ksmani@csee.wvu.edu}

## Abstract

This paper is concerned with the Negative Cost Cycle Detection problem (NCCD), i.e., the problem of checking whether a network with positive and negative costs on its arcs, has a negative cost cycle. The 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. This paper introduces a fundamentally new approach for negative cost cycle detection, which is based on exploiting the connections between NCCD and the problem of checking whether a system of difference constraints is feasible. Our algorithm, which we term as the *Stressing Algorithm*, is an incremental, comparison-based procedure which is asymptotically optimal, modulo the fastest 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 report that none exists. 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 can be extended to solve the SSSP problem, even if  $O(n)$  extra space is available. *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.*

## 1 Introduction

This paper is concerned with the problem of detecting negative cost cycles in networks. Briefly, let  $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \vec{c} \rangle$  denote a network (directed graph) with vertex set  $\mathbf{V} = \{v_1, v_2, \dots, v_n\}$ , edge set  $\mathbf{E} = \{e_1, e_2, \dots, e_m\}$  and a cost function  $\vec{c} : \mathbf{E} \rightarrow \mathbf{Z}$  that assigns an integer to the edges. The Negative Cost Cycle Detection (NCCD) problem is concerned with checking whether  $\mathbf{G}$  contains a negative cost cycle. 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 Segmentation [5] and Real-Time scheduling [15]. 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 [17]. Depending on the heuristic used to select edges to be relaxed, a wide variety 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 introduces 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.

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, followed by a proof of its correctness in Section 5. Section 6 briefly describes the problems associated with computing Single-Source Shortest paths, using the Stressing Algorithm. We conclude in Section 7, 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 [10] 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 graph. We also note that negative cost cycle detection is an integral subroutine in other network optimization problems such as the min-cost flow problem [11].

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 [17]. While the work in this paper has not been able to reduce the time bounds for NCCD, we have successfully and substantially reduced the space required for this problem.

## 3 Related Work

Approaches to NCCP 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, successive 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 NCCP, if the cost function  $\vec{c}$  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 [18] (BFCT). The Goldberg-Radzick algorithm [9] (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 graph.

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  $O(\sqrt{nm} \cdot \log N)$ . 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 [8] and [3].

- (b) Contraction-based methods - The 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 graph. 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 [16, 17].

In this paper, we focus on a new approach that is neither relaxation-based nor contraction-based; although the focus of our algorithm is the detection of a negative cost cycle, a simple modification 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. We have been unable to establish a connection between the Stressing Algorithm and the Single-Source Shortest paths problem. 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

```

Function DETECT-NEGATIVE-CYCLE ( $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \vec{c} \rangle$ )
1: for ( $i = 1$  to  $n - 1$ ) do
2:   for (each vertex  $v_i \in \mathbf{V}$ ) do
3:     Let  $c_{ij}$  denote cost of the lightest (in terms of cost) edge entering  $v_i$ .
4:     {If  $v_i$  has no incoming edge,  $c_{ij} = +\infty$ .}
5:     if ( $c_{ij} < 0$ ) then
6:       STRESS( $v_i, c_{ij}$ )
7:     end if
8:   end for
9: end for
10: if ( $\exists e_{ij} \in \mathbf{E} : c_{ji} < 0$ ) then
11:    $\mathbf{G}$  contains a negative cost cycle.
12: else
13:    $\mathbf{G}$  does not contain a negative cost cycle.
14: end if

```

**Algorithm 4.1:** The Stressing Algorithm

```

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

Algorithm 4.1 describes the details of the Stressing algorithm; at the heart of this algorithm, is the STRESS() procedure, which is applied to every vertex in each iteration of the outermost **for** loop. For notational convenience, we shall refer to this loop as **L**; further the  $i^{th}$  iteration of **L** will be denoted by **L<sub>i</sub>**.

To stress a vertex  $v_i$ , we first determine the cost of the least cost edge (say  $e_{ji}$ ) into  $v_i$ . 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  $c_{iz} = \infty$ , where  $v_z$  is a specialized “sink” vertex, such that each vertex of  $\mathbf{G}$  has an edge into it. Accordingly, if  $v_i$  is stressed, the weight of its outgoing edge is not altered. Note that  $v_z$  is a virtual vertex, in that it is never part of the actual network.
- (c) For reasons that will become apparent later, we choose the convention that the cost of edge  $e_{ij}$ , i.e., the cost of the edge from  $v_i$  to  $v_j$  is referred to as  $c_{ji}$ .

- (d) A given vertex  $v_i$  which is not stressed in the current iteration of  $\mathbf{L}$ , but could get stressed during future iterations.
- (e) In a single iteration of  $\mathbf{L}$ , 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 this iteration can be implemented in  $O(m)$  time, from which it follows that the Stressing Algorithm runs in  $O(m \cdot n)$ .
- (f) The Stressing Algorithm does not alter the topology of the network  $\mathbf{G}$ ; 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  $\mathbf{G}$  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 [17], which could take  $\Omega(n^2)$  extra space, in the worst case.

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.

## 5 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 5.1** *Let  $S' = \{\mathbf{A} \cdot \vec{x} \leq \vec{c}\}$  denote a system of difference constraints. Then  $S' \neq \phi$  if and only if the constraint system  $S = \{\mathbf{A} \cdot \vec{x} \leq \vec{c}, \vec{x} \leq \vec{0}\} \neq \phi$ .*

**Proof:** Given any polyhedron  $\mathbf{P}$ , defined purely by difference constraints, if  $\vec{x} \in \mathbf{P}$ , then so does  $\vec{x} - \vec{d}$ , where  $d$  is a positive integer and  $\vec{d} = [d, d, \dots, d]^T$ .  $\square$

Algorithm 5.1 is an incremental approach for determining the feasible solution of a constraint system  $S = \{\mathbf{A} \cdot \vec{x} \leq \vec{c}, \vec{x} \leq \vec{0}\}$ , constituted of difference constraints only.

**Function** INCREM-DIFF ( $\mathbf{A}, \vec{c}, \vec{0}$ )

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

**Algorithm 5.1:** The Incremental Algorithm for a System of Difference Constraints

Observe that, as specified, Algorithm 5.1 is a non-terminating procedure, since if the initial constraint system  $S = \phi$ , then it will recurse forever. We proceed to show that if  $S \neq \phi$ , then Algorithm 5.1 definitely terminates.  $\vec{0}$  represents the origin of the current affine space; when the algorithm is called for the first time,  $\vec{0}$  is initialized to  $\vec{0}$ .

**Definition 5.1** Given a collection of vectors  $\mathcal{S}$ , a vector  $\vec{y} \in \mathcal{S}$ , is said to be a maximal element, if  $\vec{x} \in \mathcal{S} \Rightarrow \vec{x} \leq \vec{y}$ , where the  $\leq$  relation holds componentwise.

It is not hard to see that if a set contains a maximal element, then this element is unique.

**Lemma 5.2** The set  $S = \{\mathbf{A} \cdot \vec{x} \leq \vec{c}, \vec{x} \leq \vec{0}\}$ , where  $(\mathbf{A}, \vec{c})$  is a system of difference constraints, contains a maximal element, if  $S \neq \emptyset$ .

**Proof:** If  $S$  contains a single 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 elements in  $S$  are bounded above by  $\vec{0}$ .

We say that two elements  $\vec{u}$  and  $\vec{v}$  in  $S$  are incomparable, if neither  $\vec{u} \geq \vec{v}$  nor  $\vec{v} \geq \vec{u}$ . Clearly, if every pair of elements in  $S$  is comparable, then these elements form a chain under the componentwise “ $\geq$ ” relationship and there must exist a maximal element in  $S$ . Since,  $S$  does not have a maximal element, it contains at least two elements, say,  $\vec{u}$  and  $\vec{v}$ , which are incomparable and further there is no element  $\vec{z} \in S$ , such that  $\vec{z} \geq \vec{u}, \vec{v}$ . 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  $\vec{z}$  formed by taking the componentwise maximum of  $\vec{u}$  and  $\vec{v}$ ; i.e.,  $z_i = \max(u_i, v_i)$ . We shall show that  $\vec{z} \in S$ .

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

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

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

The constraint  $l_1$  was chosen arbitrarily; we can therefore apply our analysis to every constraint. In other words setting  $z_i = \max(u_i, v_i)$ ,  $\forall i = 1, 2, \dots, n$  gives a solution to the constraint system, i.e.,  $\vec{z} \in S$ . It follows that the lattice of the elements of  $S$  under the componentwise “ $\geq$ ” relationship contains a greatest element, which is the maximal element.  $\square$

**Theorem 5.1** If  $S = \{\mathbf{A} \cdot \vec{x} \leq \vec{c}, \vec{x} \leq \vec{0}\} \neq \emptyset$ , then Algorithm 5.1 terminates by returning the maximal element of  $S$ .

**Proof:** Observe that if  $S \neq \emptyset$ , then as per Lemma 5.2, it contains a unique maximal element, say  $\vec{u} \leq \vec{0}$ . Since  $\vec{c}$  is an integral vector, we are guaranteed that  $\vec{u}$  is integral, by the theory of total unimodularity [14].

Consider the case, in which no recursive calls are made and Line 3 of Algorithm 5.1 is executed on the initial invocation; in this case,  $\vec{o} = \vec{0}$ . Note that if  $\vec{c} \geq \vec{0}$ , then  $\vec{0}$  is clearly a solution to the constraint system and hence belongs to  $S$ . Additionally,  $\vec{0}$  is the unique maximal element of  $S$ , i.e.,  $\vec{u} = \vec{o} = \vec{0}$ , and hence, the theorem holds.

We now consider the case in which one or more recursive calls are made within Algorithm 5.1.

Let  $S_0 = \{\mathbf{A} \cdot \vec{x} \leq \vec{c}, \vec{x} \leq \vec{0}\}$  denote the constraint system when INCREM-DIFF() is called for the first time. Since,  $\vec{c} \not\geq \vec{0}$ , we have a constraint of the form:  $x_i - x_j \leq c_{ij}$ ,  $c_{ij} < 0$ . 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  $S_1 = \{\mathbf{A}' \cdot \vec{x} \leq \vec{c}', \vec{x} \leq \vec{0}\}$ . Since  $x_j \leq 0$ , the constraint  $x_i - x_j \leq c_{ij}$  clearly implies that  $x_i \leq c_{ij}$ , in any solution to the constraint system. Accordingly, replacing the variable  $x_i$  by the variable  $x'_i = x_i - c_{ij}$ , where  $x'_i \leq 0$ , 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  $i^{th}$  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 “ $\geq$ ” relationship. It follows that the maximal element of  $S_0$  is translated to the maximal element of  $S_1$ . Hence, Algorithm 5.1 maintains the following invariant:  $\vec{o} \geq \vec{y}$ ,  $\forall \vec{y} \in S$ .

During each recursive call made by Algorithm 5.1, some component of the origin  $\vec{o}$  is decreased by at least unity and hence after at most  $n \cdot \|u\|_\infty$  recursive calls, we must have  $\vec{o} \leq \vec{u}$ . But, by construction,  $\vec{o} \geq \vec{y}$ , for all  $\vec{y} \in S$ , and therefore, we must have  $\vec{o} = \vec{u}$ .  $\square$

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 5.1 is in essence, solving the following linear program:

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

Given a network  $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \vec{\mathbf{c}} \rangle$ , it is a straightforward task to construct the constraint system  $S' = \{\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{c}}\}$ , such that  $S = \phi$ , if and only if  $\mathbf{G}$  contains a negative cost cycle [4]. The construction consists of two steps:

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

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 5.1, we know that  $\mathbf{G}$  contains a negative cost cycle, if and only if, the constraint system  $S = S' \wedge \vec{\mathbf{x}} \leq \vec{\mathbf{0}}$  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 5.1 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  $\mathbf{G}$  does not contain a negative cost cycle” and “If System (2) is feasible”, denote the same state of events. *The key difference between the two algorithms is that Algorithm 4.1 is 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 5.1 will not terminate, if the input system is infeasible.* Additionally, in Algorithm 4.1, we do not store the value of the origin, as is done in Algorithm 5.1. 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  $\mathbf{G}$ .

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  $\mathbf{G}$ , 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  $\mathbf{G}$  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 5.3** *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  $\Psi_z$  to denote the values 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.  $\square$

**Claim 5.1** *The  $o_i$  values in Algorithm 5.1 decrease 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 5.1.  $\square$

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 graph 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. Similarly, the set  $Z_i$  is defined as 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  $Z_i \subseteq S_i, \forall i = 0, 1, \dots, (n - 1)$ .

**Lemma 5.4** *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 5.1, we know that  $o_i$  can never increase for any vertex  $v_i$ . Thus, if System (2) is feasible and all the vertices of  $\mathbf{G}$  are stressed at least once, by Algorithm 4.1, then  $o_i < 0, \forall i$  on its termination, thereby contradicting Lemma 5.3.  $\square$

**Definition 5.2** *A vertex in the network  $\mathbf{G}$  is said to be saturated at level  $i$ , if it is stressed during  $\mathbf{L}_i$ , but never afterwards during Algorithm 4.1.*

For instance, Lemma 5.4 establishes that there is at least one vertex which is saturated at level 0.

**Lemma 5.5** *If  $S_i = \phi$ , then  $S_j = \phi$ ,  $j = (i + 1), (i + 2), \dots, (n - 1)$ . Further, System (2) is feasible and  $\mathbf{G}$  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 graph and hence the current graph 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  $j = (i + 1), \dots, (n - 1)$ . Let  $\mathbf{T} : \mathbf{A}' \cdot \vec{x} \leq \vec{c}'$  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,  $\vec{c}' \geq \vec{0}$ . Therefore,  $\mathbf{T}$  is feasible. However, as noted in Theorem 5.1, 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.  $\square$

**Theorem 5.2** *Assume that  $\mathbf{G}$  does not have a negative cost cycle. Then,*

$$(S_i \neq \phi) \rightarrow (Z_i \neq \phi), i = 0, 1, 2, \dots, n.$$

**Proof:** By convention,  $S_0 = \mathbf{V}$  and by Lemma 5.4, we know that  $Z_0 \neq \phi$ ; 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,  $\Psi$ , is maximized at a minimal face of the polyhedron  $\{\mathbf{A} \cdot \vec{x} \leq \vec{0}, \vec{x} \leq \vec{0}\}$ , say  $F$ . From the Hoffman-Kruskal theorem (See [12]), we know that  $F = \{\vec{x} : \mathbf{B} \cdot \vec{x} = \vec{c}'\}$ , where  $\{\mathbf{B} \cdot \vec{x} \leq \vec{c}'\}$  is a subsystem of the system  $\{\mathbf{A} \cdot \vec{x} \leq \vec{c}, \vec{x} \leq \vec{0}\}$ . We have already established that System (2) has a unique maximal point, i.e., the minimal face,  $F$ , corresponding to  $\Psi$  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 graph  $\mathbf{G}$ , the arcs corresponding to the difference constraints in  $\mathbf{B}$ , form a tree,  $\mathbf{B}_t$ .

We associate a vertex set  $M_i$ ,  $i = 0, 1, \dots, (n - 1)$ , 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 5.1, we know that  $o_j \leq b$  and hence  $x_j \leq b$ , 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 constraints of the basis  $\mathbf{B}$  and  $x_j \in M_0$ . 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  $x_j - x_k \leq b$  is a constraint derived by Algorithm 5.1 and hence  $x_j \leq b$  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  $x_a \in M_0$ .
- (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  $x_k \in M_1$ .

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  $M_1 = \phi$ . 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  $\mathbf{B}_1 \cdot \vec{x}_1 = \vec{b}_1$ . 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  $\mathbf{B}_1 \cdot \vec{x}_1 = \vec{b}_1$ , i.e.,  $Z_0 = \mathbf{V}$ .

We now handle the case in which  $M_1 \neq \phi$ . 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 graph, 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  $o_j \leq c'_{ja} < c_{ja}$ , 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.,  $c_{ja}$ , and stays at that value, over all future iterations. This is because 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  $x_a \in M_1$  will also be saturated at Level 1. It follows that  $M_1 = Z_1 \neq \phi$  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  $x_j \in M_1$ , now become absolute constraints, since  $x_j$  has been fixed in the current iteration.

This argument can be applied inductively for  $i = 2, 3, \dots, (n-1)$  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  $x_a \in M_{i-1}$ , or there is a constraint  $x_j - x_a = 0$ , where  $x_a$  is already in  $M_i$ .
- (ii) If  $M_i = \phi$ , the elimination of the constraints, in  $\cup_{j=1}^{i-1} M_j$  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.,  $\cup_{j=1}^{(i-1)} M_j = \mathbf{V}$ . This implies that  $Z_i = S_i = \phi$  and the theorem is true.
- (iii) If  $M_i \neq \phi$ , 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,  $M_i = Z_i \neq \phi$  and hence the theorem holds.

□

**Theorem 5.3** *If  $\mathbf{G}_i$  has a negative cost cycle, then so does  $\mathbf{G}_{i+1}$ ,  $i = 0, 1, \dots, (n-1)$ .*

**Proof:** Recall that Algorithm 4.1 does not alter the topology of the graph, but only its cost structure. 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  $R$ .



As per the mechanics of the  $\text{STRESS}()$  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  $\text{STRESS}()$  operation, the theorem follows.  $\square$

**Theorem 5.4** *The Stressing Algorithm executes Line (11 :), if and only if the input network  $\mathbf{G}$  contains a negative cost cycle.*

**Proof:** If the input network  $\mathbf{G}$  contains a negative cost cycle, then the  $\text{STRESS}()$  operations executed in lines (1 :) – (9 :) of Algorithm 4.1 are not going to change its cost, as per Theorem 5.3. Indeed, after Line (9 :) is executed for the last time,  $\mathbf{G}$  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  $\mathbf{G}$  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)  $S_i = \phi$ , for some  $i \in \{1, 2, \dots, (n-1)\}$
- (ii)  $S_i \neq \phi$ ,  $\forall i \in \{1, 2, \dots, (n-1)\}$ .

In the former case, there were no vertices to be stressed during  $\mathbf{L}_i$ , for some  $i = 0, 1, \dots, (n-1)$ . 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 graph are non-negative. This implies that Line (11 :) will not be executed.

In the latter case,  $|Z_i| \geq 1$ ,  $i = 0, 1, \dots, (n-1)$  and hence  $\cup_{i=0}^{n-1} Z_i = \mathbf{V}$ . 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.  $\square$

## 6 Computing Shortest Paths

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

Observe that only  $v_3$ ,  $v_4$  and  $v_5$  have incoming edges; assuming that the vertices are stressed in that order, then the edge costs converge at the end of the first iteration and we get the network in Figure ??.

It is not hard to check that  $\vec{o} = [0, 0, -3, -7, -1]^T$  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.

## 7 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 asymptotically optimal, modulo the fastest known algorithm for the Negative Cost Cycle 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.

In passing, we remark that any label-correcting algorithm to determine negative cost cycles requires that all vertices in the network be reachable from the specified source, whereas our algorithm does not impose this condition on the network.

An obvious avenue for extending the current research is to study an implementation suite of Stressing Algorithms, in a manner similar to the study in [3]. Additionally, we are interested in parallel versions of the Stressing Algorithm.

## 8 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, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, 1993.
- [2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 25 April 1994. Fundamental Study.
- [3] Boris V. Cherkassky, Andrew V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73:129–174, 1996.
- [4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill Book Company, Boston, Massachusetts, 2nd edition, 1992.
- [5] I.J. Cox, S.B. Rao, and Y.Zhong. Ration regions: A technique for image segmentation. In *Proceedings of the International Conference on Pattern Recognition*, pages 557–564. IEEE, August 1996.
- [6] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
- [7] J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. In IEEE, editor, *42nd IEEE Symposium on Foundations of Computer Science: proceedings: October 14–17, 2001, Las Vegas, Nevada, USA*, pages 232–241, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 2001. IEEE Computer Society Press.
- [8] Goldberg. Shortest path algorithms: Engineering aspects. In *ISAAC: 12th International Symposium on Algorithms and Computation (formerly SIGAL International Symposium on Algorithms)*, Organized by Special Interest Group on Algorithms (SIGAL) of the Information Processing Society of Japan (IPSJ) and the Technical Group on Theoretical Foundation of Computing of the Institute of Electronics, Information and Communication Engineers (IEICE)), 2001.
- [9] Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, June 1995.
- [10] C. C. Han and K. J. Lin. Scheduling real-time computations with separation constraints. *Information Processing Letters*, 12:61–66, May 1992.
- [11] M. Klein. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Managament Science*, (14):205–220, 1967.
- [12] B. Korte and J. Vygen. *Combinatorial Optimization*. Number 21 in Algorithms and Combinatorics. Springer-Verlag, New York, 2000.
- [13] P. Morris, N. Muscatella, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.

- [14] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, New York, 1987.
- [15] K. Subramani. An analysis of zero-clairvoyant scheduling. In Joost-Pieter Katoen and Perdita Stevens, editors, *Proceedings of the 8<sup>th</sup> International Conference on Tools and Algorithms for the construction of Systems (TACAS)*, volume 2280 of *Lecture Notes in Computer Science*, pages 98–112. Springer-Verlag, April 2002.
- [16] K. Subramani and L. Kovalchick. Contraction versus relaxation: A comparison of two approaches for the negative cost cycle detection problem. In et. al. Peter M. A. Sloot, editor, *Proceedings of the 3<sup>rd</sup> International Conference on Computational Science (ICCS)*, volume 2659 of *Lecture Notes in Computer Science*, pages 377–387. Springer-Verlag, June 2003.
- [17] K. Subramani and L. Kovalchick. A greedy strategy for detecting negative cost cycles in networks. *Future Generation Computer Systems*, 2004. Accepted, In Press.
- [18] Robert Endre Tarjan. *Data structures and network algorithms*, volume 44 of *CBMS-NSF Reg. Conf. Ser. Appl. Math.* SIAM, 1983.