Journal of Advances in Mathematics and Computer Science

24(1): 1-14, 2017; Article no.JAMCS.34933

Previously known as British Journal of Mathematics & Computer Science

ISSN: 2231-0851

The Cycled Shortest Path Problem: A New Perspective, And Sensitivity Analysis

Asghar Aini[1*], Kourosh Eshghi[1] and Amir Salehipour[2]

[1] *Department of Industrial Engineering, Sharif University of Technology, Azadi Avenue, 14598, Tehran, Iran.*

[2] *CARMA, The University of Newcastle NSW 2308, Australia.*

Authors' contributions

This work was carried out in collaboration between all authors. All authors equally contributed into the study. Author AA developed the major theoretical works. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/JAMCS/2017/34933

Editor(s):

(1) Huchang Liao, Business School, Sichuan University, P. R. China.

Reviewers:

(1) K. Thilagam, Pondicherry Engineering College, India.

(2) Y. Harold Robinson, Anna University, India.

(3) Vikram Puri, GNDU Regional Campus Jalandhar, India.

Complete Peer review History: http://www.sciencedomain.org/review-history/20610

Original Research Article

Received: 20th June 2017

Accepted: 30th July 2017

Published: 22nd August 2017

Abstract

Several algorithms, including the Floyd-Warshall algorithm, have been developed to calculate the shortest path between every pair of vertices in a graph (network) with cycles. This study proposes an exact algorithm, the Cascade Rectangle (CR) algorithm, for calculating the shortest paths between every pair of vertices in cycled graphs. The algorithm is developed by designing and implementing certain improvements to the available exact algorithms. In particular, the proposed algorithm has an improved computational complexity, although its worst computational complexity is $O(n^3)$. Moreover, the CR algorithm is easier to implement, which is an advantage for teaching and learning purposes. In addition to this, we introduce a new concept, the transposition matrix, which has important applications in sensitivity analysis and re-optimization of the shortest path networks. An example illustrates the CR algorithm and the new concept of transposition matrix.

**Corresponding author: E-mail: ainiasghar@ie.sharif.edu;*

Keywords: The cycled shortest path problem; Cascade Rectangle algorithm; transposition matrix; sensitivity analysis; shortest path network re-optimization.

1 Introduction

The shortest path problem is a fundamental and well-known problem in operations research. The problem is to obtain a path between every two vertices (nodes) of a graph such that the sum of the weights (cost, distance, time, etc.) of its connecting edges is minimized ([1], [2], [3]). The shortest path problem has many real-world applications including in the transportation and telecommunication networks. One common application is to find the quickest route through a road network. In this application, vertices represent geographical locations, and edges that connect the vertices represent possible routes between geographical locations. The weight of an edge typically reflects traveling time or distance. Fig. 1 depicts two examples of such a road network with four vertices (geographical locations). Here, vertex 1 is the source (the depot or the starting point), and vertex 4 is the sink (the destination). The figure on the left has five arcs, which represent one-way routes between the geographical locations. The figure on the right illustrates a cycled graph with 10 arcs, which in fact, are five two-way routes. Such a graph represents two-way road networks. As you may notice, along an arc the traveling time or distance may differ depending on the direction of the arc. According to the figure on the left, the shortest path from the source to the sink is through vertex 3 and has a cost of 5.

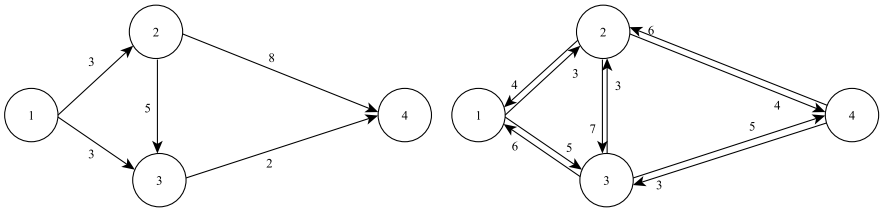

Fig. 1. Two graphs with four vertices. The graph on the left, which represents a one-way road network, has no cycles, and the graph on the right, which represents a two-way road network has cycles

Generally, the shortest path problem is categorized into cases without cycles (the left one in Fig. 1), and cases with cycles (the right one in Fig. 1). There are algorithms for both cases where an optimal solution is guaranteed [4]. In the cases with cycles there is not a source and a sink. Thus, every vertex can be a source or sink.

Several algorithms exist to obtain the shortest path and the associated route between every pair of vertices in a graph. In a graph with cycles, the first algorithms were proposed by [5] and [6]. A "cascade" algorithm was developed by [7]. Hu has improved the usual matrix method for obtaining all shortest paths in a graph [8]. Considering integer weights on the graph's edges, an efficient algorithm to obtain all pairs of the shortest paths was proposed by [9]. Aini and Salehipour developed a new algorithm [10], which is based on some enhancements to the algorithm of [5] and [6]. For a more detailed discussion about the shortest path problem we refer the interested reader to [11].

Although, the shortest path problem belongs to the computational complexity class $\mathcal{P}$ (see [11] for a discussion), its variants including existence of negative cycles belong to the complexity class $\mathcal{NP}$-Hard [12]. In a directed graph with cycles, an approximation algorithm for the shortest path problem with real values for the edges' weights was proposed by [13]. Peng et al. developed an

algorithm by applying several improvements to the Dijkstra's algorithm [14]. Their algorithm has a reduced computational complexity. Several algorithms were also proposed in a study by [15]. The literature on the variants of the problem is also very rich. For example, in [16], the authors developed an algorithm to determine all shortest paths in Sierpiński graphs. [17] compared several integer programming formulations for the shortest path problem, and provided a polyhedral study of the formulations. In a note by [18] an optimality condition on Bellman-Ford-Moore classical algorithm is discussed. Researchers also solved variants of the problem by using heuristic and meta-heuristic algorithms [19, 20].

In this work, we study the shortest path problem with cycles. We develop a novel exact algorithm, the Cascade Rectangle (CR) algorithm, for the shortest path problem with cycles, which is the first contribution of this study. The algorithm is an improvement of the Farbey et al.'s, Hu's, Floyd-Warshall, and Aini and Salehipour's algorithms. The proposed CR algorithm is based on a graphical concept, which makes its understanding and illustration easy to follow; this is an advantage, in particular, for teaching and learning purposes. The second contribution of this study is on sensitivity analysis of the shortest path graphs. Particularly, we introduce the transposition matrix that analyze the impact of graphs vertices and arcs on the optimal solution, and has direct applications in re-optimization of the shortest path graphs. An example illustrates the implementation of the CR algorithm and the application of the transposition matrix.

The remainder of this study is organized as follows. Section 2 defines the problem, and provides the mathematical notations and an integer programming formulation. Section 3 discusses the developed Cascade Rectangle (CR) algorithm. This section also includes the complexity analysis of the CR algorithm. Section 4 introduces the transposition matrix, which is a novel concept on the sensitivity analysis of the shortest path graphs. In Section 5, we illustrate a cycled graph to explain the operation of the CR algorithm, and the computation of the transposition matrix. The paper ends with a few conclusions.

2 The Problem Statement

Let $G = (V, E, C)$ be a graph, where $V = \{1, \ldots, n\}, |V| = n$ is a set of vertices (nodes), $E = \{e_{ij} : i, j \in V\}, |E| = m$ is a set of edges, and $C = \{c_{ij} \in \mathbb{R}^+ : i, j \in V\}, |C| = m$ is a set of edges' weights (for example, traveling cost, time, distance, priority, etc.). An edge $e_{ij} := (i, j)$ connects two vertices i and $j \in V$. Generally speaking, if for at least one pair of i and j, $c_{ij} \neq 1$ (which indeed implies a weighted graph), then, G is called a network, and is denoted by $N = (V, E, C)$. A directed graph $D = (V, A, C)$ is similar to G with the difference that the edges have directions, and hence, they are called arcs, and can be denoted by the set A. In a directed graph D, $e_{ij} \neq e_{ji}$, whereas in an undirected graph G, $e_{ij} = e_{ji}$.

A *walk* on D is a sequence of vertices $V' \subseteq V$ and arcs $A' \subseteq A$. A walk with unique vertices and arcs is called a *path*. A closed walk is called a *cycle*.

Let $D = (V, A, C)$ be a directed graph with a set of vertices V, a set of arcs A, and a set of arcs' weights $C = \{c_{ij} \in \mathbb{R}^+ : i, j \in V\}$. The shortest path problem on graph (network) D is to obtain the shortest path between every pair of vertices $s, i \in V$, where $s := 1$ is the source vertex, also known as the depot or starting vertex.

The shortest path problem on D can be formulated as an integer program (IP) with decision variables $x_{ij} \in \{0, 1\}, \forall e_{ij} \in A$. The variable takes 1 if arc e_{ij} is used to reach to vertex j, from vertex i, and 0 otherwise. The constraints ensure the solution is indeed a path [11].

$$\text{Minimize } z = \sum_{e_{ij} \in A} c_{ij} x_{ij} \tag{1}$$

subject to

$$\sum_{i:e_{ij} \in A} x_{ij} - \sum_{j:e_{ji} \in A} x_{ji} = \begin{cases} n-1 & \text{for} \quad i := s \\ -1 & \text{for} \quad i \in N - \{s\} \end{cases} \tag{2}$$

$$x_{ij} = \{0, 1\}, \forall e_{ij} \in A \tag{3}$$

Equation (1) minimizes the total cost of travel. Notice that traveling time, distance, etc. may easily be represented by the traveling cost. Constraints (2) are flow constraints and ensure there is a path between every pair of vertices. Constraints (3) ensure decision variables are binary.

3 The Cascade Rectangle Algorithm

This section presents the major contribution of this study, which is an exact algorithm for the cycled shortest path problem by modifications on Farbey et al.'s [7], Hu's [8], the well-known Floyd-Warshall's, and Aini-Salehipour's [10] algorithms. The major benefit of the developed algorithm lies in its reduced computational complexity, in compared to the state-of-the-art algorithms.

3.1 Preliminaries

Assume a directed graph $D = (V, A, C)$. Also, assume that at least one cycle exists on D. The Floyd-Warshall algorithm ([5] and [6]) is probably the most well-known and one of the best algorithms for obtaining the shortest path between every two vertices $i, j \in V$. The Floyd-Warshall algorithm recursively computes two square $n \times n$ matrices $\mathbf{D}^S_{n\times n}$ and $\mathbf{R}^S_{n\times n}$ for $S = 0, \ldots, n$. The matrix $\mathbf{D}^n_{n\times n}$ holds the shortest path weights (costs, distances, etc.), and matrix $\mathbf{R}^n_{n\times n}$ holds the shortest routes (the complete path, including start and end vertices), between every two arbitrary vertices i and j. Notice that $\mathbf{D}^S_{n\times n}$ and $\mathbf{R}^S_{n\times n}$ are calculated $n+1$ times, where each matrix has $n^2 - n$ entities (because both matrices are square, and we do not compute the diagonal). Details of this algorithm can be found in [5], [6], and [10].

In order to obtain the shortest path and weight between every pair of vertices $i, j \in V$, the Cascade Rectangle (CR) algorithm only calculates four $n \times n$ matrices $\mathbf{D}^S$ and $\mathbf{R}^S$, where $S = 0, 1$ (note that in the Floyd-Warshall algorithm $S = 0, \ldots, n$). The stage $S = 0$ is the *initial* stage of the CR algorithm, and the stage $S = 1$ is the *final* stage. As before, matrix $\mathbf{D}^1_{n\times n}$ holds the shortest weights and matrix $\mathbf{R}^1_{n\times n}$ holds the shortest routes, between every two arbitrary vertices of V. Finally, the CR algorithm derives the shortest path routes, i.e. $\mathbf{R}^S_{n\times n}$, only when $\mathbf{D}^S_{n\times n}$ is calculated.

We picked the name "Cascade Rectangle" for the algorithm because all computations and operations may be performed via drawing a set of rectangles. As we will see in Section 3.3, the Cascade Rectangle algorithm is easy to implement as well as easy to be understood; this is an advantage, both for implementation as well as for teaching and learning purposes.

3.2 The operation of the Cascade Rectangle algorithm

The Cascade Rectangle (CR) algorithm has two stages $S = 0, 1$. In stage $S = 0$, the algorithm defines two square $n \times n$ matrices $\mathbf{D}^0_{n\times n}$ and $\mathbf{R}^0_{n\times n}$. More precisely, the initial weights and routes (paths) are calculated at stage $S = 0$, and are stored in $\mathbf{D}^0_{n\times n}$ and $\mathbf{R}^0_{n\times n}$. The shortest weights and routes are updated at stage $S = 1$ through *forward* and *backward* calculations, and are stored

in $\mathbf{D}^1_{n\times n}$ and $\mathbf{R}^1_{n\times n}$. This is because for an entity $d^1_{ij} \in \mathbf{D}^1_{n\times n}$, $2(n-2)$ computations must be performed in order to update d^1_{ij}. More precisely, for every d^1_{ij}, $n-2$ operations are needed (later, we show how those $n-2$ computations may visually be illustrated as a set of $n-2$ rectangles). This is an improvement to the previous algorithms (e.g. [7, 8]), where n operations are needed.

The CR algorithm is on the basis of the algorithms of [7], [8], [5], [6], and [10]. In particular, it transforms the underlying mathematical equations (Equation (6)) into rectangles, as well as reducing the number of calculations of the shortest weights and routes matrices. Hence, its correctness is followed by the correctness of those algorithms. Algorithm 1 illustrates the CR algorithm.

Algorithm 1: The Cascade Rectangle (CR) algorithm to obtain shortest paths and weights between every two arbitrary vertices of a cycled graph $D = (V, A, C)$.

Step 1. Assume a directed graph $D(V, A, C)$ with a set of vertices $V, |V| = n$, a set of arcs $A, |A| = m$, and a set of arcs' edges $C, |C| = m$ are given. In stage $S = 0$, define two $n \times n$ matrices $\mathbf{D}^0_{n\times n}$ and $\mathbf{R}^0_{n\times n}$.

Step 2a. Calculate matrix $\mathbf{D}^0_{n\times n}$. The row i and column j of $\mathbf{D}^0_{n\times n}$ is denoted by d^0_{ij} and represents the weight (distance, cost, time, etc.) in order to travel from vertex i to vertex j. Equation (4) calculates $d^0_{ij} \in \mathbf{D}^0_{n\times n}$.

$$d^0_{ij} = \begin{cases} c_{ij}, & \text{if vertex } i \text{ is connected to vertex } j; \\ \infty, & \text{if vertex } i \text{ is not connected to vertex } j; \\ 0, & \text{if } i = j; \end{cases} \tag{4}$$

Step 2b. Calculate matrix $\mathbf{R}^0_{n\times n}$. The row i and column j of $\mathbf{R}^0_{n\times n}$ is denoted by r^0_{ij} and represents the route (an arc) in order to travel from vertex i to vertex j. Equation (5) for $S = 0$, calculates $r^0_{ij} \in \mathbf{R}^0_{n\times n}$.

$$r^S_{ij} = \begin{cases} j, & \text{if } d_{ij} \neq 0 \text{ or } d_{ij} \neq \infty; \\ -, & \text{otherwise;} \end{cases} \tag{5}$$

Step 3a. In stage $S = 1$, pre-process matrices $\mathbf{D}^1_{n\times n}$ and $\mathbf{R}^1_{n\times n}$. The diagonal elements of $\mathbf{D}^1_{n\times n}$ and $\mathbf{R}^1_{n\times n}$ are from $\mathbf{D}^0_{n\times n}$ and $\mathbf{R}^0_{n\times n}$. That is, $d^1_{ij} \in \mathbf{D}^1_{n\times n} = 0$, and $r^1_{ij} \in \mathbf{R}^1_{n\times n} = -, \forall i = j$.

Step 3b. Calculate matrix $\mathbf{D}^1_{n\times n}$.
Forward calculation: forward calculation starts from d^1_{12} and ends in $d^1_{n,n-1}$. To compute d^1_{ij}, $n-2$ rectangles are needed (see Fig. 2 for drawing a rectangle). Equation (6) shows how the value of the rectangle in Fig. 2 may mathematically be expressed. Indeed, both the rectangle's value and Equation (6) calculate $d^1_{ij} \in \mathbf{D}^1_{n\times n}$ (notice that super index 0 and 1 in Equation (6) refer to $S = 0, 1$).

$$d^1_{ij} = \min(d^0_{ij}, d^0_{ik} + d^0_{kj}), \forall k, k \neq i, k \neq j \tag{6}$$

Backward calculation: backward calculation starts from $d^1_{n,n-1}$ and ends in d^1_{12}. The calculation is similar to that of the forward calculation.

Step 3c. Calculate matrix $\mathbf{R}^1_{n\times n}$:
Unlike the algorithms of [8], [7], [5], and [6], there is no calculations in order to derive matrix $\mathbf{R}^1_{n\times n}$. Note that, the calculation of entities of $\mathbf{R}^1_{n\times n}$ is followed by matrix $\mathbf{D}^1_{n\times n}$. Thus, if $d^1_{ij} \in \mathbf{D}^1_{n\times n}$ does not change, then $r^1_{ij} \in \mathbf{R}^1_{n\times n}$ will not change as well. Otherwise, r^1_{ij} will change, and the change depends on the zeros of the diagonal (of the associated rectangle). If the associated rectangles lead to the same value for Equation (6), then $r^1_{ij} \in \mathbf{R}^1_{n\times n}$ will be derived from row i and columns associated with the diagonal of those "equal" rectangles.

3.3 Generating rectangles

At stage $S = 1$ of the CR algorithm (Step 3b in Algorithm 1), for every $d^1_{ij} \in \mathbf{D}^1_{n\times n}$, $n-2$ rectangles are drawn. This is because in order to compute d^1_{ij} we cannot use the zeros of row i and column j. The rectangles are drawn in a way that $d^1_{kk} = 0$ of the diagonal is on one corner and d^1_{ij} is on the opposite corner (these two are always opposite to each other). The other two corners are d^1_{ik} and d^1_{kj}. This is illustrated in Fig. 2. In the figure, d_{ij} refers to the entity associated with row i and column j of matrix $\mathbf{D}^1_{n\times n}$, i.e. d^1_{ij}. Each rectangle is identified by its top left (row i and column j) and bottom right (row k and column k) corners: $M_{(i,j),(k,k)}$.

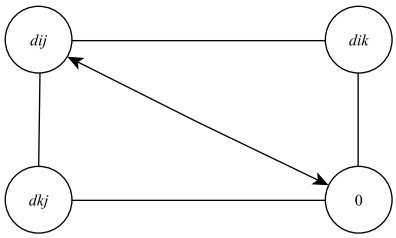

Fig. 2. Constructing a rectangle in the Cascade Rectangle algorithm, where '0' refers to d_{kk} (on the diagonal at stage $S = 1$). Note that how four entities d_{ij}, d_{ik}, d_{kj}, and $d_{kk} = 0$ of $\mathbf{D}^1_{n\times n}$ form a rectangle

Based on the concept of rectangles, the value of $d^1_{ij}, i \neq j$ in the matrix $\mathbf{D}^1_{n\times n}$ is calculated by Equation (6). Notice that if d^1_{ij} changes, the new value is computed through Equation (6), and $\mathbf{D}^0_{n\times n}$ and $\mathbf{D}^1_{n\times n}$ are updated accordingly. If $d^1_{ij} \leq d^0_{ik}$ or $d^1_{ij} \leq d^0_{kj}$, then $d^1_{ij} \leq d^0_{ik} + d^0_{kj}$; this may easily be performed in the CR algorithm.

3.4 Correctness proof of the Cascade Rectangle algorithm

In this section, we provide the correctness proof of the Cascade Rectangle (CR) algorithm. Recall that the major advancements in the CR algorithms are 1) utilizing rectangles to replace Equation (6), and that includes $n-2$ operations for stages $S = 1$, against n operations of the Farbey et al.'s, Hu's, and Floyd-Warshal's algorithms, and 2) advancement in deriving matrices $\mathbf{R}^0_{n\times n}$ and $\mathbf{R}^1_{n\times n}$. For this reason, the correctness of the CR algorithm is followed by first proving that the rectangles indeed illustrate Equation (6), and then proving the correctness of new calculation of matrix $\mathbf{R}^S_{n\times n}, S = 0, 1$.

Theorem 1. *The matrix $\mathbf{D}^1_{n\times n}$ may be derived by drawing a set of $(n^2 - n) \times (n - 2)$ rectangles.*

Proof. Let us calculate $d^1_{ij}, i \neq j$. For this reason, the CR algorithm draws a rectangle with the zeros on the diagonal of matrix $\mathbf{D}^0_{n\times n}$ (remember that the diagonal only includes zeros). An example of one such rectangle is illustrated in Fig. 2. Here, the value of the rectangle may be derived by its corners: $d^1_{ij} = \min(d^0_{ij} + 0, d^0_{ik} + d^0_{kj}) = \min(d^0_{ij}, d^0_{ik} + d^0_{kj})$, and this equals Equation (6). Equation (6) is the one used in [7] and [8] to derive matrix $\mathbf{D}^S_{n\times n}$. ■

Indeed, Theorem 1 proves that Equation (6) may be represented as rectangles and vice versa. Now, in Theorem 2 we prove an alternative way to derive matrix $\mathbf{R}^S_{n\times n}, S = 0, 1$.

Theorem 2. *The entities of* $\mathbf{R}^S_{n\times n}, S = 0, 1$ *depend on* $\mathbf{D}^S_{n\times n}, S = 0, 1$*, and hence, can be calculated by Equation* (5)*.*

Proof. The proof includes two parts. The first part includes stage $S = 0$, and the second part includes stages $S = 1, \ldots, n$. Let us start by the stage $S = 0$. The Floyd-Warshall algorithm calculates r_{ij} by Equation (7):

$$r_{ij} = \begin{cases} -, & \text{if } i = j \\ j, & \text{if there is an arc from } i \text{ to } j \\ -, & \text{if there is not an arc from } i \text{ to } j \end{cases} \tag{7}$$

Notice that we do not directly calculate r^0_{ij}, instead, in the CR algorithm we calculate r^0_{ij}, and obtain the values by analyzing d^0_{ij}. According to Equation (4), if d^0_{ij} has a value, then, we already know that there is an arc connecting vertex i to vertex j, thus, $r^0_{ij} = j$. Similarly, if $d^0_{ij} = \infty$, then, we know that there is no arc connecting vertex i to vertex j; hence, $r^0_{ij} = -$.

For stages $S = 1, \ldots, n$, the Floyd-Warshall algorithm calculates r_{ij} by Equation (8):

$$r_{ij} = \begin{cases} j, & \text{if } i = k \text{ or } i = j \text{ or } i = k \\ j, & \text{if } i \neq k \neq j \text{ and } d_{ij} \leq d_{ik} + d_{kj} \\ k, & \text{if } i \neq k \neq j \text{ and } d_{ij} > d_{ik} + d_{kj} \end{cases} \tag{8}$$

By looking into Equation (8), one may recognize that the calculation is very similar to that of Equation (6). Indeed, if Equation (6) leads to d_{ij}, then, we do not need to verify this when calculating r^1_{ij} (this is, however, verified in Equation (8)). Thus, $r^1_{ij} = j$. Similarly, if either Equation (4) or Equation (6) leads to $d_{ik} + d_{kj}$, we may safely have $r^1_{ij} = k$. ■

3.5 Time complexity of the Cascade Rectangle algorithm

Theorem 3. *The worst time complexity of the Cascade Rectangle (CR) algorithm is* $O(n^3)$*, where* n *is the number of vertices.*

Proof. Given the graph $D = (V, A, C)$, where $|V| = n$, there are $n^2 - n$ operations in order to calculate matrix $\mathbf{D}^0_{n\times n}$. In addition to this, the number of computations for matrix $\mathbf{D}^1_{n\times n}$ is bounded by $2(n^2 - n)$, as the number of forward and backward computations are equal ([7, 8]). Thus, there are $2(n^2 - n) + (n^2 - n)$ computations for matrices $\mathbf{D}^0_{n\times n}$ and $\mathbf{D}^1_{n\times n}$.
Moreover, in order to compute every element of $\mathbf{D}^1_{n\times n}$, $n - 2$ operations are needed (in the form of generating rectangles, see Fig. 2, and also, Equation (6) for equivalent calculations). This implies that the total number of operations to derive matrices $\mathbf{D}^0_{n\times n}$ and $\mathbf{D}^1_{n\times n}$ is $2(n^2-n)(n-2)+(n^2-n)$. Similarly, it is not difficult to show that the route matrices of $\mathbf{R}^0_{n\times n}$ and $\mathbf{R}^1_{n\times n}$ require $2(n^2 - n)$ calculations. Thus, the total number of calculations of the CR algorithm is $2(n^2-n)(n-2)+3(n^2-n)$. Putting all those together result in a worst time complexity of $O(n^3)$ for the CR algorithm. ■

Notice that although the worst time complexity of the CR algorithm remains $O(n^3)$, it has indeed a better performance than the Floyd-Warshall's, Farbey et al.'s, and Hu's algorithms. This is mainly because the CR algorithm calculates $\mathbf{R}^S_{n\times n}, S = 0, 1$ based on $\mathbf{D}^S_{n\times n}, S = 0, 1$. This has been illustrated in Fig. 3. In this figure, we analyzed four different sized graphs including 20, 50, 70, and 100 vertices (illustrated by the horizontal axis), and calculated the total number of operations of each algorithm (illustrated by the vertical axis). As the figure shows, the worst performance, among three, belongs to the Hu's and Farbey et al.'s algorithms. Also, while the performance of the CR and the Floyd-Warshall algorithms are very close, the former has a slightly better performance as measured in terms of less number of computational operations.

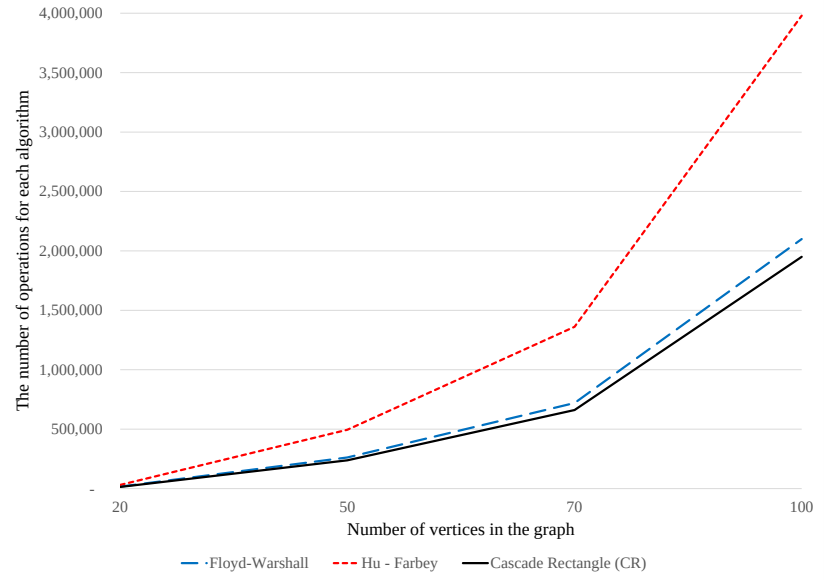

Fig. 3. Comparing the worst performance of the Floyd-Warshall, Hu, Farbey, and Cascade Rectangle (CR) algorithms on instances with sizes from 20 to 100 vertices. Notice that the worst performance of all three algorithms is $O(n^3)$, where n is the number of vertices. However, the average performance of the CR algorithm is better than the Floyd-Warshall, and Hu's and Farbey's algorithms, as evidenced by this figure

4 The Transposition Matrix

As discussed earlier, the Cascade Rectangle (CR) algorithm obtains an optimal solution, i.e. the shortest weights and the associated routes for the shortest path problem with cycles. The optimal solution is in the form of matrix $\mathbf{D}^1_{n\times n}$, the optimal weights, and matrix $\mathbf{R}^1_{n\times n}$, the optimal routes.

Given the route matrix of $\mathbf{R}^1_{n\times n}$, we can derive a new concept, the *transposition* matrix. The elements of this matrix show the number of traversals of arcs in an optimal solution. Having the transposition matrix, we can define the value of an arc in the graph. Following this, the transposition matrix has a variety of applications in the sensitivity analysis and re-optimization of the shortest path graphs.

Similar to matrices $\mathbf{D}^S_{n\times n}$ and $\mathbf{R}^S_{n\times n}$, the transposition matrix $\mathbf{T}$ is a square $n \times n$ matrix. The number of traversals of an arc $e_{ij} \in A$ may be denoted by $t_{ij} \in \mathbf{T}_{n\times n}$ and we call it *arc transposition*

number. In other words, t_{ij} shows the number of traversals between vertices i and j in the optimal solution; if we remove arc e_{ij}, then t_{ij} traversals will be impacted. Thus, the greater the value of t_{ij}, the more crucial role the arc e_{ij} plays in the optimal solution. Equation (9) shows how elements of the transposition matrix is computed.

$$t_{ij} = \begin{cases} -, & \text{if } i = j \\ \sum_{j=1, i\neq j, r^1_{ij}=i}^{j=n} r^1_{ij} + \sum_{i=1, i\neq j, r^1_{ij}=j}^{i=n} r^1_{ij}, & \text{if } r^1_{ij} = j \\ 0, & \text{Otherwise} \end{cases} \tag{9}$$

Similarly, value of an arc is called *arc transposition value*, and is denoted by v_{ij}. This value is presented as percentage, and is the ratio of t_{ij} to the total number of traversals in the graph. Assume the summation over all elements of matrix $\mathbf{T}_{n\times n}$ is τ, i.e. $\sum_{t_{ij}\in \mathbf{T}_{n\times n}} t_{ij} = \tau$. Then, the transposition value of an arc, i.e. $v_{ij}, \forall e_{ij} \in A$ can be derived by Equation (10).

$$v_{ij} = \frac{t_{ij}}{\tau} \times 100 \tag{10}$$

Given t_{ij} and v_{ij}, we may conceptualize the importance of every arc in the graph. As a result, we may find the redundant arcs and remove them.

We may define similar concepts to that of the arc transposition number (t_{ij}), and the arc transposition value (v_{ij}) for every vertex $i \in V$. In fact, every vertex i has connections to at most $n-1$ vertices, and those $n-1$ vertices may be connected to vertex i as well. Thus, the total number of traversals of vertex i to all other vertices is at most $2(n-1)$. If we add the number of times that a vertex i is *intermediate*, then we may have the total number of traversals of vertex i. This is, though, on one condition, and that is, we should not have an $r^1_{ij} = \infty$; otherwise, we subtract the number of times an ∞ appears from $2(n-1)$. Let t_i and v_i denote the transposition number and value of vertex $i \in V$. Equation (11) derives t_i and Equation (13) derives v_i.

$$t_i = 2(n-1) + \eta_i, r^1_{ik} \neq \infty \tag{11}$$

where $\eta_i =$ is the number of intermediate vertices to vertex i. Equation (12) calculates η_i.

$$\eta_i = \sum_{k=1, k\neq i}^{n} \sum_{i=1, i\neq k}^{n} r^1_{ik} \tag{12}$$

Notice that the intermediate vertices to vertex i, i.e. η_i, can be extracted from $\mathbf{R}^1_{n\times n}$ by counting all $r^1_{ij}, i \neq j$.

$$v_i = \frac{t_i}{(2n(n-1) + \sum_{i=1}^{n} \eta_i)} \times 100, r^1_{ik} \neq \infty \tag{13}$$

In fact, v_i represents the value of vertex i over all optimal traversals in the graph. Thus, if we remove vertex i, $v_i\%$ of the graph, in the optimal solution, will be impacted. Similar to v_{ij}, the greater the value of v_i, the more important the role of vertex i is in the optimal traversals. Notice that these concepts have important applications in the sensitivity analysis and re-optimization of the shortest path graphs because they represent impact of every vertex and arc on the optimal solution.

Now we proceed to Section 5 with one illustrative example to explain the operation of the CR algorithm, and to establish the concept of the transposition matrix.

5 An Illustrative Example

In this section, we give an example to clarify the Cascade Rectangle (CR) algorithm and the concept of the transposition matrix. The purpose of the example is to discuss, in more details, the operation of the algorithm and explain the important role of the transposition matrix. So its purpose is not to analyze and verify the computational complexity or the performance of the Cascade Rectangle (CR) algorithm. A detailed discussion on this has been brought in Section 3.4.

Consider the directed graph D in Fig. 4 with four vertices and 10 arcs, where $V = \{1, 2, 3, 4\}$, and $A = \{e_{12}, e_{21}, e_{13}, e_{31}, e_{23}, e_{32}, e_{24}, e_{42}, e_{34}, e_{43}\}$. We aim to compute the shortest path and weight between every pair of vertices of D.

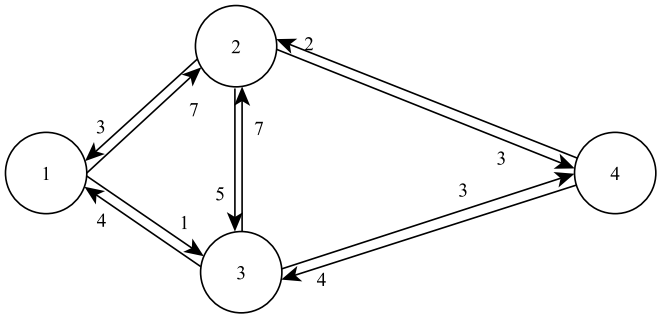

Fig. 4. The graph of the example that includes four vertices and 10 arcs

According to Step 1 of Algorithm 1, we defined $\mathbf{D}^0_{4\times4}$ and $\mathbf{R}^0_{4\times4}$.

In Step 2, elements of $\mathbf{D}^0_{4\times4}$ (Step 2a), i.e. d^0_{ij}, and $\mathbf{R}^0_{4\times4}$ (Step 2b), i.e. r^0_{ij}, are calculated by Equation (4) and Equation (5):

$$d^0_{ij} = \begin{cases} c_{ij}, & \text{if vertex } i \text{ is connected to vertex } j \\ \infty, & \text{if vertex } i \text{ is not connected to vertex } j \\ 0, & \text{if } i = j \end{cases}$$

and,

$$r^0_{ij} = \begin{cases} j, & \text{if } d^0_{ij} \neq 0 \text{ or } d^0_{ij} \neq \infty \\ -, & \text{otherwise} \end{cases}$$

This results in $\mathbf{D}^0_{4\times4} = \begin{bmatrix} 0 & 7 & 1 & \infty \\ 3 & 0 & 5 & 3 \\ 4 & 7 & 0 & 3 \\ \infty & 2 & 4 & 0 \end{bmatrix}$ and, $\mathbf{R}^0_{4\times4} = \begin{bmatrix} - & 2 & 3 & - \\ 1 & - & 3 & 4 \\ 1 & 2 & - & 4 \\ - & 2 & 3 & - \end{bmatrix}$.

Step 3 of Algorithm 1 calculates $\mathbf{D}^1_{4\times4}$ and $\mathbf{R}^1_{4\times4}$ matrices. More precisely, Step 3a illustrates that the elements on the diagonal of $\mathbf{D}^1_{4\times4}$ and $\mathbf{R}^1_{4\times4}$ are taken from their counterparts in matrices $\mathbf{D}^0_{4\times4}$ and $\mathbf{R}^0_{4\times4}$. Step 3b explains the forward calculations, which starts from d^1_{12} and ends in d^1_{43}. Here, in order to calculate every $d^1_{ij} \in \mathbf{D}^1_{4\times4}, \forall i, j \in V, i \neq j$, $n - 2 = 2$ rectangles are needed. Note that removing those $d^1_{ij}, i = j$, we will be left with 12 elements. Thus, the total number of operations are 24 (or 12×2).

In order to draw rectangles for d^1_{ij}, the 0 of the diagonal is on one corner and d^1_{ij} is on the opposite corner (see Section 3.3 and Fig. 2). Recall that we specify a rectangle $M_{(i,j),(k,k)}$ with its start

corner on row i and column j and its end corner on row k and column k. For example, if 0 is on row 3 and column 3 and d^1_{12} is on row 1 and column 2, then, the rectangle is denoted by $M_{(1,2),(3,3)}$.

In order to calculate d^1_{12}, two rectangles $M_{(1,2),(3,3)}$ and $M_{(1,2),(4,4)}$ are formed: $M_{(1,2),(3,3)} = \begin{bmatrix} 7 & 1 \\ 0 & 5 \\ 7 & 0 \end{bmatrix}$ and $M_{(1,2),(4,4)} = \begin{bmatrix} 7 & 1 & \infty \\ 0 & 5 & 3 \\ 7 & 0 & 3 \\ 2 & 4 & 0 \end{bmatrix}$.

Following Equation (6), d^1_{12} is calculated over two diagonals of each rectangle. Hence, $d^1_{12} = \min(d^0_{12} = 7, d^0_{31}+d^0_{13} = 7+1, d^0_{42}+d^0_{14} = 2+\infty) = 7$. Continuing this, $d^1_{13} = \min(1, 7+5, 4+\infty) = 1$, and $d^1_{14} = \min(\infty, 7+3, 1+3) = 4$.

Following Step 3c of Algorithm 1, as $d^1_{12} = d^0_{12}$ and $d^1_{13} = d^0_{13}$, thus $r^1_{12} = r^0_{12} = 2$ and $r^1_{13} = r^0_{13} = 3$. On the other hand, $d^1_{14} = \min(\infty, 7+3, 1+3) = 4$, and $d^1_{14} \neq d^0_{14}$, which implies $r^1_{14} \neq r^0_{14}$. Remember that the change in r^1_{ij} depends on the zeros of the diagonal. Moreover, $M_{(1,4),(2,2)} = \begin{bmatrix} 7 & 1 & \infty \\ 0 & 5 & 3 \end{bmatrix}$ and $M_{(1,4),(3,3)} = \begin{bmatrix} 1 & \infty \\ 5 & 3 \\ 0 & 3 \end{bmatrix}$. Following these two rectangles, the new value of d^1_{14} is because of the rectangle $M_{(1,4),(3,3)}$; more precisely, because of the entity d^1_{33}, thus, $r^1_{14} = 3$. Finally, at the end of Step 3b or the forward calculation, we have matrices $\mathbf{D}^1_{4\times4} = \begin{bmatrix} 0 & 7 & 1 & 4 \\ 3 & 0 & 4 & 3 \\ 4 & 5 & 0 & 3 \\ 5 & 2 & 4 & 0 \end{bmatrix}$ and $\mathbf{R}^1_{4\times4} = \begin{bmatrix} - & 2 & 3 & 3 \\ 1 & - & 1 & 4 \\ 1 & 4 & - & 4 \\ 2 & 2 & 3 & - \end{bmatrix}$.

Now we must proceed to Step 3b, the backward calculation, which starts from d^1_{43} and ends is d^1_{12}. For instance, to calculate d^1_{43}, the two rectangles are $M_{(2,2),(4,3)} = \begin{bmatrix} 0 & 4 \\ 5 & 0 \\ 2 & 4 \end{bmatrix}$ and $M_{(1,1),(4,3)} = \begin{bmatrix} 0 & 7 & 1 \\ 3 & 0 & 4 \\ 4 & 5 & 0 \\ 5 & 2 & 4 \end{bmatrix}$. Thus, $d^1_{43} = \min(4, 2+4, 5+1) = 4$. Let us calculate d^1_{12}. Two rectangles are $M_{(1,2),(3,3)} = \begin{bmatrix} 7 & 1 \\ 0 & 4 \\ 5 & 0 \end{bmatrix}$ and $M_{(1,2),(4,4)} = \begin{bmatrix} 7 & 1 & 4 \\ 0 & 4 & 3 \\ 5 & 0 & 3 \\ 2 & 4 & 0 \end{bmatrix}$; thus, $d^1_{12} = \min(7, 5+1, 2+4) = 6$. Notice that d^1_{12} has changed and the change is because of the rectangle $M_{(1,2),(3,3)}$, which is associated with $d^1_{33} = 0$. Interestingly, in the other rectangle, i.e. $M_{(1,2),(4,4)}$, we also have a value of 6, which is associated with $d^1_{44} = 0$. Because $r^1_{13} = r^0_{14} = 3$, hence, $r^1_{12} = 3$. Finally, $\mathbf{D}^1_{4\times4}$ and $\mathbf{R}^1_{4\times4}$ matrices, after both forward and backward calculations, are $\mathbf{D}^1_{4\times4} = \begin{bmatrix} 0 & 6 & 1 & 4 \\ 3 & 0 & 4 & 3 \\ 4 & 5 & 0 & 3 \\ 5 & 2 & 4 & 0 \end{bmatrix}$ and

$\mathbf{R}^1_{4\times4} = \begin{bmatrix} - & 3 & 3 & 3 \\ 1 & - & 1 & 4 \\ 1 & 4 & - & 4 \\ 2 & 2 & 3 & - \end{bmatrix}$. Having those two matrices in hand, the shortest weights and routes between every two vertices i and j of graph D is then available.

In order to derive the transposition matrix, we utilize matrix $\mathbf{R}^1_{n\times n}$. Also, recall that the diagonal of matrix $\mathbf{T}_{n\times n}$ follows the diagonal of $\mathbf{R}^1_{n\times n}$. Thus, the elements on the diagonal are "$-$". In order to derive the remaining elements, we follow Equation (9). For example, to calculate t_{12}, we look at row 1 in matrix $\mathbf{R}^1_{4\times4}$ and count the number of elements that equals 2 plus the number of elements in column 2 that equals 1. For t_{13}, we count the number of elements that equals 3 in row 1 plus the number of elements in column 3 that equals 1. If an element $r^1_{ij} \neq j$, then $t_{ij} = 0$. More precisely, $\mathbf{T}_{4\times4} = \begin{bmatrix} - & 0 & 3+1 & 0 \\ 2+1 & - & 0 & 1+0 \\ 1+0 & 0 & - & 2+1 \\ 0 & 2+1 & 1+0 & - \end{bmatrix} = \begin{bmatrix} - & 0 & 4 & 0 \\ 3 & - & 0 & 1 \\ 1 & 0 & - & 3 \\ 0 & 3 & 1 & - \end{bmatrix}$.

From our previous discussion in Section 4, the elements of matrix $\mathbf{T}_{n\times n}$ represent arc transposition numbers. For example, $t_{21} = 3$ implies that arc e_{21} has been used 3 times in the optimal solution, i.e. in three shortest paths. In other words, if we remove this arc, three shortest paths are impacted. This implies we may need to perform a re-optimization. The remaining arc transposition numbers along with their transposition values are as the followings.

$t_{12} = 0, t_{13} = 4, t_{14} = 0,$
$t_{21} = 3, t_{23} = 0, t_{24} = 1,$
$t_{31} = 1, t_{32} = 0, t_{34} = 3,$
$t_{41} = 0, t_{42} = 3, t_{43} = 1.$
Also, $t = \sum_{t_{ij}\in T} = 16$; hence,
$v_{12} = 0/16 = 0\%, v_{13} = 4/16 = 25\%, v_{14} = 0/16 = 0\%,$
$v_{21} = 3/16 = 18.75\%, v_{23} = 0/16 = 0\%, v_{24} = 1/16 = 6.25\%,$
$v_{31} = 1/16 = 6.25\%, v_{32} = 0/16 = 0\%, v_{34} = 3/16 = 18.75\%,$
$v_{41} = 0/16 = 0\%, v_{42} = 3/16 = 18.75\%, v_{14} = 1/16 = 6.25\%.$

For example, if arc e_{13} is removed, then 25% of the graph is affected. However, if arc e_{23} is removed, then, the graph is not affected. Thus, this arc is redundant, or in fact, it does not exist in the shortest paths (an optimal solution).

In order to derive the transposition number and value of every vertex, we need to count the intermediate vertices. For this reason, we may count the number i associated with vertex i in all columns of $\mathbf{R}^1_{4\times4}$ but the column i. For example, in order to see how many intermediate vertices are to vertex 1, we do not consider column 1, and count entities that equal 1 in the remaining columns of $\mathbf{R}^1_{4\times4}$. We can see that the number of intermediate vertices to vertex 1 is 1, to vertex 2 is 1, to vertex 3 is 2, and to vertex 4 is 0. Hence, the total number of intermediate vertices is 4. Moreover, as we do not have an ∞ in $\mathbf{R}^1_{4\times4}$, then every vertex has a total of $2(4-1) = 6$ traversals. The transposition number and value of each vertex are then:

$t_1 = 1+6 = 7, t_2 = 1+6 = 7, t_3 = 2+6 = 8, t_4 = 1+6 = 7,$ and
$v_1 = \frac{1+6}{5+2\times4\times3} = 7/29 = 24.14\%,$
$v_2 = \frac{1+6}{5+2\times4\times3} = 7/29 = 24.14\%,$
$v_3 = \frac{2+6}{5+2\times4\times3} = 8/29 = 27.58\%,$
$v_4 = \frac{1+6}{5+2\times4\times3} = 7/29 = 24.14\%.$

For example, if we remove vertices 1, 2, or 4, 24.14% of the optimal graph traversals will be impacted; for the case of vertex 3, the amount is 27.58%.

6 Conclusion

In this study, we introduced an enhanced algorithm for calculating the shortest path in graphs with cycles. The developed algorithm, the Cascade Rectangle (CR) algorithm, improves on the Floyd-Warshall's, Farbey et al.'s, Hu's and Aini-Salehipour's algorithms, four of the best available algorithms for tackling this problem. The Floyd-Warshall algorithm and the CR algorithm have exactly the same performance in deriving $\mathbf{D}^0_{n\times n}$ and $\mathbf{R}^0_{n\times n}$ matrices. For the stages $S \geq 1$, however, the CR algorithm derives the matrices more quickly due to the reduced number of operations. In the second part of the study, we introduced a novel concept of transposition matrix that has applications in the shortest path re-optimization and sensitivity analysis. In particular, we introduced the arc transposition number and value, which quantify the importance of each arc on the optimal solution, and the vertex transposition number and value, which quantify the importance of each vertex on the optimal solution. As a future research, we are working on expanding the transposition matrix, and its added value in quantifying the role of arcs and vertices in an optimal solution.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Bellman RE. On a routing problem. Quarterly of Applied Mathematics. 1958;16(1):87-90.

[2] Dijkstra EW. A note on two problems in connection with graphs. Numeriskche Mathematik. 1959;1:269-271.

[3] Ford DR, Fulkerson DR. Flows in networks. Princeton University Press, Princeton, NJ, USA; 1962.

[4] Giorgio Gallo, Stefano Pallottino. Shortest path algorithms. Annals of Operations Research. 1988;13(1):1-79.

[5] Robert W. Floyd. Algorithm 97: shortest path. Communications of the ACM. 1962;5(6):345.

[6] Stephen Warshall. A theorem on boolean matrices. Journal of the ACM. 1962;9(1):11-12.

[7] Farbey BA, Land AH, Murchland JD. The cascade algorithm for finding all shortest distances in a directed graph. Management Science. 1967;14(1):19-28.

[8] Hu TC. Revised matrix algorithms for shortest paths. SIAM Journal on Applied Mathematics. 1967;5(1):207-218.

[9] Zvi Galil, Oded Margalit. All pairs shortest paths for graphs with small integer length edges. Journal of Computer and System Sciences. 1997;54(2):243-254.

[10] Asghar Aini, Amir Salehipour. Speeding up the floyd-warshall algorithm for the cycled shortest path problem. Applied Mathematics Letters. 2012;25(1):1-5.

[11] Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin. Network Flows: Theory, Algorithms, and applications. Prentice-Hall, Inc., Upper Saddle River, NJ, USA; 1993.

[12] Stefan Hougardy. The Floyd-Warshall algorithm on graphs with negative cycles. Information Processing Letters. 2010;110(8):279-281.

[13] Raphael Yuster. Approximate shortest paths in weighted graphs. Journal of Computer and System Sciences. 2012;78(2):632-637.

[14] Wei Peng, Xiaofeng Hu, Feng Zhao, Jinshu Su. A fast algorithm to find all-pairs shortest paths in complex networks. Procedia Computer Science. 2012;9:557-566.

[15] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Frank Neumann, Madeleine Theile. More effective crossover operators for the all-pairs shortest path problem. Theoretical Computer Science. 2013;471:12-26.

[16] Andreas M. Hinz, Caroline Holz auf der Heide. An efficient algorithm to determine all shortest paths in sierpiski graphs. Discrete Applied Mathematics. 2014;177:111-120.

[17] Leonardo Taccari. Integer programming formulations for the elementary shortest path problem. European Journal of Operational Research. 2016;252(1):122-130.

[18] Stasys Jukna, Georg Schnitger. On the optimality of bellman-ford-moore shortest path algorithm. Theoretical Computer Science. 2016;628:101-109.

[19] Bingüler A, Hande Erol, Bulkan, Serol. New Variable Neighborhood Search Structure for Travelling Salesman Problems. British Journal of Mathematics & Computer Science. 2015;6(5):422-434.

[20] Oloduowo Ameen, Babalola Akande Hakeem, Daniel Aruleba Kehindel. Solving network routing problem using artificial intelligent techniques. British Journal of Mathematics & Computer Science. 2016;17(3):1-9.

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
http://sciencedomain.org/review-history/20610