DOI: 10.22098/jhs.2026.17764.1106

Journal of Hyperstructures, 15 (2) (2026), 233-248

Journal Homepage: https://jhs.uma.ac.ir/

Research Paper

SET-THEORETIC FORM OF WARSHALL'S ALGORITHM FOR COMPUTING TRANSITIVE CLOSURE OF A BINARY RELATION

ABOUTORAB POURHAGHANI[1] AND SEID MOHAMMAD ANVARIYEH[2,*]

[1]Department of Mathematical Sciences, Yazd University, Yazd, Iran, pourhaghani@stu.yazd.ac.ir

[2]Department of Mathematical Sciences, Yazd University, Yazd, Iran, anvariyeh@yazd.ac.ir

ARTICLE INFO

Article history:
Received: 4 July 2025
Accepted: 6 June 2026
Communicated by Irina Cristea

Keywords:
Binary relation
Transitive closure
Warshall's Algorithm

MSC:
05C09; 05C15; 05C76

ABSTRACT

In this paper, we investigate about the transitive closure of a binary relation. We present the algorithms for computing the transitive closure in matrix form and set-theoretic form, as well as their complexity. First, we recall a simple algorithm (Algorithm 3.1) and the Warshall's algorithm(Algorithm 4.1), along with their complexity, to obtain the matrix of transitive closure. Then, we introduce the Algorithm 3.2 and Warshall's algorithm (Algorithm 4.2) in set-theoretic form, instead of matrix form, as well as their complexity. Finally, the Maple codes of the newly introduced algorithms are provided along with an example of how to use them in Maple.

1. Introduction

One of the most important and applicable issues in mathematics is finding closures of a binary relation. For instance, in fundamental relations in algebraic hyperstructures (such as the β relation and its transitive closure, β^*), we need to compute the transitive closure of a relation defined on an algebraic hyperstructure [4]. The closure of a binary relation has a general definition. But three types of closures are more important and practical than the rest, namely: *reflexive*, *symmetric* and *transitive* closures.

Finding the reflexive and symmetric closures of a binary relation R on a set A is easily obtained as $R \cup \{(a,a)|a \in A\}$ and $R \cup \{(b,a)|(a,b) \in R\}$ respectively, but obtaining the

*Address correspondence to Seid Mohammad Anvariyeh; Department of Mathematical Sciences, Yazd University, Yazd, Iran, E-mail: anvariyeh@yazd.ac.ir.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.
Copyright © 2026 The Author(s). Published by University of Mohaghegh Ardabili.

transitive closure is not as easy as the previous two closure. Look at the Example 1.1(Ref. [6, p. 629]).

Example 1.1. Let $R = \{(1,3),(1,4),(2,1),(3,2)\}$ be a relation on the set $A = \{1,2,3,4\}$. We can see the relation R is not transitive. We have $(1,2),(2,3),(2,4),(3,1) \notin R$. By adding these pairs to R we will not have a transitive relation, because the resulting relation contains the pairs $(3,1)$ and $(1,4)$ but does not contain the pair $(3,4)$.

Example 1.1 shows that finding the transitive closure is not as simple as the other two closures. Therefore, finding the transitive closure will be more complicated. Theorem 3.11 shows a primitive method that we can use to find the transitive closure of a binary relation. Also, theorem 3.11 shows that finding the transitive closure requires more computations than the other two closures.

Algorithm 3.1 and Algorithm 3.2, based on Theorem 3.11, compute the transitive closure of a binary relation. Default assumption in this paper is that sets are finite. Algorithm 3.1 deals with the matrix form of a relation, while Algorithm 3.2 deals directly with members of the relation. Algorithm 3.3 is a detailed form of Algorithm 3.2, so that it can be easily implemented in mathematical software or programming languages.

Considering Algorithm 3.1 or Algorithm 3.2 shows that computing the transitive closure of a binary relation requires a large amount of computations, and this has caused efforts to find a faster algorithm. The result of these efforts is an algorithm known as Warshall's algorithm. Warshall's algorithm is used in two equivalent topics: finding the shortest path in a graph and finding the transitive closure of a binary relation. This algorithm was discovered by Roy[7](1959) and Warshall [9](1962) ([3, pp. 330-331 and 353-356] and [10, 1]). Our main goal in this paper is to find the transitive closure of a binary relation, so we exclude the shortest path problem. The best-known transitive closure algorithm is Warshall's algorithm, which is given in many textbooks (e.g. [2, p. 19] and [8, p. 263]).

There are two methods for computing the transitive closure of a binary relation. In first method, first the zero–one matrix of the desired relation is created. Then the matrix of transitive closure of the relation is obtained. Finally, the transitive closure of the relation is obtained from the matrix. In second method, by using set operators, such as adding member to a set and union of sets, the transitive closure will be computed directly. The advantage of the latter method is that the transitive closure is computed directly without the need for matrix computations.

Based on research done by the authors of the paper, the existing algorithms obtaining the transitive closure of a relation use first method, and indeed, no algorithm was found based on second method yet.

In this paper, we try to compute the transitive closure of a binary relation using second method and present Algorithm 3.2 and set-theoretic form of Warshall's algorithm (Algorithm 4.2). These two algorithms are new and presented in two simple and detailed modes as well as their order of time complexity based on RAM model. Algorithm 3.2 is the set-theoretic form of Algorithm 3.1, and Algorithm 4.2 is the set-theoretic form of Warshall's algorithm (Algorithm 4.1).

Since most mathematical software and programming languages strongly support set-theoretic instructions (such as algebraic operations on sets, including union, intersection, etc.) and execute these instructions at an appropriate speed, and since such instructions

allow for the direct programming of set-theoretic pseudocode in these environments; without the need for converting sets to matrices and vice versa; it seems necessary to develop algorithms for computing closures of binary relations in set-theoretic form. This is a key motivation for this paper. The purpose of the algorithms presented in this paper is to enable mathematical programmers to implement algorithms involving closure applications more conveniently within their own code.

2. Preliminaries

In this section, we recall some definitions and properties of binary relations. The content of this section is mainly adopted from [6, 5] with slight changes. Preliminary definitions in set theory, group theory, ring theory and matrices are assumed to be known, but some special definitions are mentioned. It should be noted that the proofs of the propositions and theorems of this section and the following sections are presented in [6, pp. 624-636] and therefore omitted, but the proofs of the new propositions and theorems are presented.

2.1. **Binary Relations And Properties.** Let A and B be non-empty sets. Every subset of $A \times B$ is defined as a *binary relation* from A to B.

Since we use only binary relations, if there is no ambiguity, we remove the word binary and only use the word relation instead of binary relation.

A *relation R on a set A* is defined as $R \subseteq A \times A$. The relation R on a set A is called *transitive* if from $(a, b), (b, c) \in R$ follows that $(a, c) \in R$, for every $a, b, c \in A$.

Let R and S are relations that $R \subseteq A \times B$ and $S \subseteq B \times C$. The *composite* of R and S is define as follow:

$$S \circ R = \{(a, c) \in A \times C \mid \exists b \in B \text{ such that } (a, b) \in R \text{ and } (b, c) \in S\}.$$

This definition yields the composition of relations is associative, that is, if R, S and T are three relations on A, then we have $T \circ (S \circ R) = (T \circ S) \circ R$. Now, let $A = \{a_1, ..., a_n\}$ be a finite set of cardinality n, and let $\mathcal{R}(A) = \{R \mid R \subseteq A \times A\}$ denotes the set of all binary relations on A. Since the composition of relations is associative, it yields $(\mathcal{R}(A), \circ)$ is a semigroup.

Let R be a relation on the set A. We define the *powers* R^n, for $n = 1, 2, 3, \ldots$, as follow:

$$R^1 = R$$
$$R^{n+1} = R^n \circ R.$$

2.2. **Representing Relations.** We can use several methods to represent relations on finite sets. In this paper, we use set of ordered pairs of a relation and zero–one matrices for this purpose.

2.2.1. *Required Definitions In Matrices.* In this paper, we let $\mathcal{D} = \{0, 1\}$.

Let $x, y \in \mathcal{D}$. We define Boolean operations $\wedge$, $\vee$, and $\neg$ as follow:

$$x \wedge y = \begin{cases} 1 & \text{if } x = y = 1 \\ 0 & \text{o.w.} \end{cases} \qquad x \vee y = \begin{cases} 1 & \text{if } x = 1 \text{ or } y = 1 \\ 0 & \text{o.w.} \end{cases} \qquad \neg x = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x = 1 \end{cases}$$

The structure $(\mathcal{D}, \vee, \wedge)$ is a commutative semiring: $(\mathcal{D}, \vee, 0)$ is a commutative monoid with identity element 0, $(\mathcal{D}, \wedge, 1)$ is a monoid with identity element 1, the operation $\wedge$ is

distributive over $\vee$ on both sides, and 0 is an absorbing element for $\wedge$, i.e. $a \wedge 0 = 0 \wedge a = 0$. Since the element 1 admits no additive inverse with respect to $\vee$ (because $1 \vee 0 \neq 0$ and $1 \vee 1 \neq 0$), this structure is not a ring, but only a semiring. The algebraic structure $(\mathcal{D}, \vee, \wedge)$ is called a *Boolean semiring*.

Now we consider the set of all $m \times n$ matrices with entries from the semiring $(\mathcal{D}, \vee, \wedge)$.

Definition 2.1. A matrix $\mathbf{A}$ that all its entries are in $\mathcal{D}$ is called a *zero–one matrix* or *Boolean matrix*. In other words, we have $\mathbf{A} = [a_{ij}]_{m\times n} \ : \ a_{ij} \in \mathcal{D} = \{0, 1\}$.

The all-zero matrix $\bar{0}$ and the all-one matrix $\bar{1}$ serve as the least and greatest elements, respectively.

Let $m, n \in \mathbb{N}$. The $\mathcal{M}_{m,n}(\mathcal{D})$ means that the set of all $m \times n$ zero–one matrices endowed with operations $\bar{\wedge}$, $\bar{\vee}$ and $\bar{\neg}$ defined componentwise on the matrix entries. In other words, let $\mathbf{A}, \mathbf{B} \in \mathcal{M}_{m,n}(\mathcal{D})$ and $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$. Then we define

$$\mathbf{A} \bar{\vee} \mathbf{B} = [a_{ij} \vee b_{ij}]_{m\times n} \qquad \mathbf{A} \bar{\wedge} \mathbf{B} = [a_{ij} \wedge b_{ij}]_{m\times n} \qquad \bar{\neg}\mathbf{A} = [\neg a_{ij}]$$

For $m = n$, we denote $\mathcal{M}_{n,n}(\mathcal{D})$ by $\mathcal{M}_n(\mathcal{D})$.

Definition 2.2. Let $\mathbf{A} \in \mathcal{M}_{m,k}(\mathcal{D})$ and $\mathbf{B} \in \mathcal{M}_{k,n}(\mathcal{D})$. The map $\odot$ is defined as

$$\begin{cases} \odot : \mathcal{M}_{m,k}(\mathcal{D}) \times \mathcal{M}_{k,n}(\mathcal{D}) \to \mathcal{M}_{m,n}(\mathcal{D}) \\ \mathbf{A} \odot \mathbf{B} = [c_{ij}]_{m\times n}, \\ c_{ij} = \bigvee_{t=1}^{t=k} (a_{it} \wedge b_{tj}) \end{cases}$$

This means that $\mathbf{A} \odot \mathbf{B}$ is obtained via the usual row-by-column matrix multiplication, using the Boolean operations from $\mathcal{D}$. The map $\odot$ is called the *Boolean product* of $\mathbf{A}$ and $\mathbf{B}$.

Since the product of matrices is associative, $\odot$ is also associative on $\mathcal{M}_n(\mathcal{D})$, and thus $(\mathcal{M}_n(\mathcal{D}), \odot)$ is a semigroup. In the following subsection we will examine the relationship between $(\mathcal{R}(A), \circ)$ and $(\mathcal{M}_n(\mathcal{D}), \odot)$.

Now, we provide an example about the Boolean product of matrices.

Example 2.3. Let $\mathbf{A} = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ and $\mathbf{B} = [\begin{smallmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{smallmatrix}]$. We have

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} (1\wedge1)\vee(0\wedge0) & (1\wedge1)\vee(0\wedge1) & (1\wedge0)\vee(0\wedge1) \\ (0\wedge1)\vee(1\wedge0) & (0\wedge1)\vee(1\wedge1) & (0\wedge0)\vee(1\wedge1) \\ (1\wedge1)\vee(0\wedge0) & (1\wedge1)\vee(0\wedge1) & (1\wedge0)\vee(0\wedge1) \end{bmatrix} = \begin{bmatrix} 1\vee0 & 1\vee0 & 0\vee0 \\ 0\vee0 & 0\vee1 & 0\vee1 \\ 1\vee0 & 1\vee0 & 0\vee0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

Definition 2.4. Let $\mathbf{A}$ be a square zero–one matrix and n be a positive integer. The *Boolean power* of $\mathbf{A}$ is defined as follows:

$$\mathbf{A}^{[n]} = \underbrace{\mathbf{A} \odot \mathbf{A} \odot \mathbf{A} \odot \cdots \odot \mathbf{A}}_{n \text{ times}}$$

This operation is well-defined, because the Boolean product of matrices is associative. Also we define

$$\mathbf{A}^{[0]} = \mathbf{I}_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}_{n\times n}$$

Example 2.5. Let $\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$. We have

$$\mathbf{A}^{[0]} = \mathbf{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{A}^{[1]} = \mathbf{A} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \qquad \mathbf{A}^{[2]} = \mathbf{A}^{[1]} \odot \mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$
$$\mathbf{A}^{[3]} = \mathbf{A}^{[2]} \odot \mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \mathbf{A}^{[4]} = \mathbf{A}^{[3]} \odot \mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \mathbf{A}^{[5]} = \mathbf{A}^{[4]} \odot \mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \bar{1}.$$

It is easy to check that $\mathbf{A}^{[n]} = \mathbf{A}^{[5]} = \bar{1}$ for every integer $n \geq 5$.

2.2.2. *Representing Relations Using Matrices.* Let R be a relation from the set $A = \{a_1, a_2, \ldots, a_m\}$ to the set $B = \{b_1, b_2, \ldots, b_n\}$. We can represent the relation R by the matrix $\mathbf{M}_R = [m_{ij}]_{m \times n}$, that

$$m_{ij} = \begin{cases} 1 & \text{if } (a_i, b_j) \in R \\ 0 & \text{if } (a_i, b_j) \notin R. \end{cases}$$

This representation depends on the orderings for A and B. When $A = B$ we use the same ordering for A and B.

Now we determine the matrix of composition of two relations.

Proposition 2.6. *Let R and S are relations that $R \subseteq A \times B$ and $S \subseteq B \times C$. Let A, B, and C are finite sets and have m, n, and p elements respectively. Then we have*

$$\mathbf{M}_{S \circ R} = \mathbf{M}_R \odot \mathbf{M}_S.$$

Now, consider the following natural map between $(\mathcal{R}(A), \circ)$ and $\mathcal{M}_n(\mathcal{D})$:

$$\begin{cases} \theta : \mathcal{R}(A) \to \mathcal{M}_n(\mathcal{D}) \\ R \mapsto \mathbf{M}_R = [m_{ij}]_{n \times n} \ ; \quad i, j \in \{1, ..., n\} \end{cases} , \quad m_{ij} = \begin{cases} 1 & \text{if } (a_i, a_j) \in R, \\ 0 & \text{o.w.} \end{cases}$$

By Proposition 2.6 we conclude $\theta(S \circ R) = \mathbf{M}_R \odot \mathbf{M}_S$ and $\theta^{-1}(\mathbf{M}_R \odot \mathbf{M}_S) = S \circ R$. Therefore, $\theta(S \circ R) \neq \theta(S) \circ \theta(R)$. It means that map θ is not a homomorphism. On the other hand, it is obvious map θ is bijective. So we conclude that $(\mathcal{R}(A), \circ)$ is not isomorphic to $(\mathcal{M}_n(\mathcal{D}), \odot)$, but corresponds to it.

Although the map θ is not an isomorphism, its bijectivity enables us to convert relation R into $\mathbf{M}_R$ and vice versa, a property sufficient for the continuation of this paper.

Example 2.7. Let $A = \{a, b, c\}$, $R = \{(a, a), (a, c), (b, a), (b, b)\}$ and $S = \{(a, b), (b, c), (c, a), (c, c)\}$. Then $S \circ R = \{(a, a), (a, b), (a, c), (b, b), (b, c)\}$ and $R^2 = R \circ R = \{(a, a), (a, c), (b, a), (b, b), (b, c)\}$. On the other hand $\mathbf{M}_R = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $\mathbf{M}_S = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$. Then we have

$$\mathbf{M}_{S \circ R} \overset{2.6}{=\!=} \mathbf{M}_R \odot \mathbf{M}_S = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{M}_{R^2} = \mathbf{M}_R^{[2]} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Thus $\theta(S \circ R) = \theta(R) \odot \theta(S)$ and $\theta(R^2) = \theta(R \circ R) = \theta(R) \odot \theta(R)$.

Remark 2.8. Let R be a relation on the finite set A, and $\mathbf{M}_R$ be its corresponding zero–one matrix. Since $\circ$ on $\mathcal{R}$ and $\odot$ on $\mathcal{M}_n(\mathcal{D})$ are associative, then we have $R^n \circ R = R \circ R^n$ and $\mathbf{A}^{[n]} \odot \mathbf{A} = \mathbf{A} \odot \mathbf{A}^{[n]}$, for $n \in \mathbb{N}$.

The remark 2.8 and map θ enable us to provide algorithms, that are presented for zero–one matrices, similarly for their corresponding binary relations on a set A.

3. Transitive Closure Of Relations

In this section, first, we define the transitive closure of a relation R based on definition of transitive property. The transitive closure of a relation R is defined as the smallest transitive relation S that contains R. Other closures are defined similarly.

Although there is not necessarily the closure of a relation R on a set A with respect to arbitrary property **P** exists, but the three famous closures, i.e. reflexive, symmetric, and transitive closures, can be computed. However, obtaining the transitive closure of a relation is more complicated than two other closures. In the rest of this section, we try to present the algorithms for computing the transitive closure.

Definition 3.1. A *path* from a to b in a relation R on a set A is a finite sequence of elements a, x_1, x_2, ... , x_{n-1} and b with $(a, x_1) \in R$, $(x_1, x_2) \in R$, ... , and $(x_{n-1}, b) \in R$. This path is denoted by $ax_1x_2 \cdots x_{n-1}b$ and has length n. A path of length $n \geq 1$ that begins and ends at the same elements is called a *circuit* or *cycle*.

Definition 3.2. Let R is a relation on a set A. If $ax_1x_2 \cdots x_{n-1}b$ is a path of length $n \geq 1$ in R from a to b, its *interior vertices* is defined as $x_1, x_2, \ldots, x_{n-1}$, that is, all the vertices of the path that occur somewhere other than as the first and last vertices in the path. We denote interior vertices of a path by $V_{a,b}$. Hence we have $V_{a,b} = \{x_1, x_2, \ldots, x_{n-1}\}$.

Theorem 3.3. *Let R be a relation on a set A. There is a path of length $n \geq 1$, from a to b if and only if $(a, b) \in R^n$.*

Example 3.4. Suppose that $R = \{(a, b), (a, c), (b, a), (b, d), (c, b), (c, d)\}$ is a relation on the set $A = \{a, b, c, d\}$. The $acbd$ is a path from a to d of length three, therefore $(a, d) \in R^3$. The aba is a cycle of length two, so $(a, a) \in R^2$.

Definition 3.5. Let R be a relation on a set A. The *connectivity relation* R^∞ consists of the pairs (a, b) such that there is a path of length at least one from a to b in R.

By Theorem 3.3, the next lemma can be deduced.

Lemma 3.6. *Let R be a relation on a set A. Then* $R^\infty = \bigcup_{i=1}^{\infty} R^i$.

Example 3.7. According to the Example 3.4, we have $(a, d) \in R^\infty$ and also $(a, a) \in R^\infty$.

Notation 3.8. For simplicity we denote the transitive closure R by R^*.

Theorem 3.9. *Let R be a relation on a set A. Then we have $R^* = R^\infty$.*

Theorem 3.9 says $R^* = R^\infty$ and so we should compute every power of R, to obtain R^*. But this is not desirable. Lemma 3.10 give a bound for R^∞ when A is a finite set.

Lemma 3.10. *Let A be a finite set with $|A| = n$ and R be a relation on A. If there is a path of length at least one in R from a to b, then there is a path with length not exceeding n without any repeated interior vertices.*

By Lemma 3.6, Theorem 3.9 and Lemma 3.10, we have the following important theorems.

Theorem 3.11. *Let A be a finite set with n elements and R be a relation on A. Then we have*

$$R^* = R^\infty = R \cup R^2 \cup R^3 \cup \cdots \cup R^n = \bigcup_{i=1}^{n} R^i.$$

Theorem 3.12. *Let $\mathbf{M}_R$ be the zero–one matrix of the relation R on a set A with $|A| = n$. Then we have*

$$\mathbf{M}_{R^*} = \mathbf{M}_R \bar{\vee} \mathbf{M}_R^{[2]} \bar{\vee} \mathbf{M}_R^{[3]} \bar{\vee} \cdots \bar{\vee} \mathbf{M}_R^{[n]}.$$

Theorem 3.12 is a basis for an algorithm for computing $\mathbf{M}_{R^*}$. This algorithm is presented in Algorithm 3.1.

Algorithm 3.1: Computing the Transitive Closure Matrix ($\mathbf{M}_{R^*}$).

```
1 Input: M_R (zero–one n × n matrix)
2 Output: M_R* (zero–one n × n matrix for R*)
3 X := M_R
4 Y := M_R
5 for i = 2 to n do
6 |   X := X ⊙ M_R
7 |   Y := Y ∨̄ X
8 end for
9 return Y
```

Based on [6, p. 634], the order of Algorithm 3.1 is $O(n^4)$ bit operations.

Example 3.13. Let $R = \{(a,b),(b,a),(b,c),(c,d)\}$ and $A = \{a,b,c,d\}$. Since $|A| = 4$, we must calculate $\mathbf{M}_R$, $\mathbf{M}_{R^{[2]}}$, $\mathbf{M}_{R^{[3]}}$ and $\mathbf{M}_{R^{[4]}}$. $\mathbf{M}_{R^*}$ and R^* is obtained using Algorithm 3.1 or Theorem 3.12 as follows:

(1) $\mathbf{M}_R = \begin{bmatrix} 0&1&0&0\\ 1&0&1&0\\ 0&0&0&1\\ 0&0&0&0 \end{bmatrix}$,

(2) $\mathbf{M}_R \bar{\vee} \mathbf{M}_{R^{[2]}} = \begin{bmatrix} 1&1&1&0\\ 1&1&1&1\\ 0&0&0&1\\ 0&0&0&0 \end{bmatrix}$,

(3) $\mathbf{M}_R \bar{\vee} \mathbf{M}_{R^{[2]}} \bar{\vee} \mathbf{M}_{R^{[3]}} = \begin{bmatrix} 1&1&1&1\\ 1&1&1&1\\ 0&0&0&1\\ 0&0&0&0 \end{bmatrix}$,

(4) $\mathbf{M}_R \bar{\vee} \mathbf{M}_{R^{[2]}} \bar{\vee} \mathbf{M}_{R^{[3]}} \bar{\vee} \mathbf{M}_{R^{[4]}} = \begin{bmatrix} 1&1&1&1\\ 1&1&1&1\\ 0&0&0&1\\ 0&0&0&0 \end{bmatrix} = \mathbf{M}_{R^*}$.

Hence $R^* = \{(a,a),\ (a,b),\ (a,c),(a,d),\ (b,a),(b,b),\ (b,c),(b,d),\ (c,d)\}$.

In an analogous way, Theorem 3.11 is a basis for an algorithm to compute R^*, while only the matrix form of algorithms in this area is presented in scientific resources. The Algorithm 3.2 is designed, similar to the Algorithm 3.1, for set-theoretic form of it. In Algorithm 3.1, we have used Remark 2.8 to compose the relation R with $\mathbf{X}$ from the right side (line 6).

To evaluate the time complexity of the Algorithm 3.2, we do in the same way as the Algorithm 3.1, with the difference that here the basic special operations are adding members and checking the membership of a member in the set, instead of bit operations in the matrix ($\wedge$ and $\vee$). It should be noted that basic operations in the set have different time complexities according to the type of definition of the set in mathematical software. In some cases, the set is defined as a linked list, while in others it is defined as an (dynamic) array. The membership of a member in a set can be checked by a for loop or a hash table (depending on

Algorithm 3.2: Computing the Transitive Closure of a Relation (R^*).

```
1  Input: Binary relation R ⊆ A × A, |A| = n
2  Output: Transitive closure: R*
3  X := R
4  Y := R
5  for i = 2 to |A| do
6  |  X := X ∘ R
7  |  Y := Y ∪ X
8  end for
9  return Y
```

the implementation of the membership function). Also adding a member can be done using index operators or pointers, according to the type of set definition, that is done in a certain and fixed time.

First, we rewrite the Algorithm 3.2 in terms of basic operations of the sets to find the complexity. We have Algorithm 3.3.

Algorithm 3.3: Computing the Transitive Closure of a Relation (R^*)(detailed form)

```
1  Input: Binary relation R ⊆ A × A, |A| = n
2  Output: Transitive closure: R*
3  X := R
4  Y := R
5  for i = 2 to |A| do
6  |  T := ∅
7  |  for (a, b) ∈ X do
8  |  |  for (c, d) ∈ R do
9  |  |  |  if b = c and (a, d) ∉ T then
10 |  |  |  |  T := T ∪ {(a, d)}    /* This step computes R^{i+1} from X = R^i and
   |  |  |  |     R */
11 |  |  |  end if
12 |  |  end for
13 |  end for
14 |  X := T                                          /* Now X = R^{i+1} */
15 |  Y := Y ∪ T                    /* This step computes R ∪ R^2 ∪ ··· ∪ R^i */
16 end for
17 return Y
```

To analyse Algorithm 3.3, we use the RAM model for this. The details of this topic are given in [1] and [8]. In short, in the RAM model each instruction or data access takes a constant amount of time, even indexing into an array [1]. To find time complexity of Algorithm 3.3, we calculate time complexity of each step separately and finally add them together. The execution steps of the algorithm start from line number 3. We assume that assignment operations, mathematical operators and simple "if" conditions require a specific and fixed time to execute, which depends on the hardware. We denote these specific times with C_1, C_2 and ..., which is called the cost of running the algorithm. If a number of commands are repeated (i.e. loop block), the execution time will be equal to the product of the number of repetitions in the execution time of the commands. The total execution time

of Algorithm 3.3 is equal to the sum of the execution time of the pieces of the algorithm, which is denoted by $T(n)$ with $n = |A|$.

TABLE 1. Complexity of Algorithm 3.3

Line	Cost	Time	Total time of the step in algorithm
3	C_1	1	$C_1 \times 1$
4	C_2	1	$C_2 \times 1$
5	C_3	$\|A\|$	$C_3 \times \|A\|$
6	C_4	$\|A\| - 1$	$C_4 \times (\|A\| - 1)$
7	C_5	$\|\mathbf{X}\| + 1$	$C_5 \times ((\|A\| - 1)(\|\mathbf{X}\| + 1))$
8	C_6	$\|R\| + 1$	$C_6 \times ((\|A\| - 1)\|\mathbf{X}\|(\|R\| + 1))$
9	C_7	1	$C_7 \times ((\|A\| - 1)\|\mathbf{X}\|\|R\|)$
10	C_8	1 or $\|\mathbf{X}\|$	$C_8 \times ((\|A\| - 1)\|\mathbf{X}\|\|R\|)$ or $C_8 \times ((\|A\| - 1)\|\mathbf{X}\|^2\|R\|)$
14	C_9	$\|\mathbf{X}\|$ or $\|\mathbf{X}\|\|\mathbf{Y}\|$	$C_9 \times \|A\|\|\mathbf{X}\|$ or $C_9 \times \|A\|\|\mathbf{X}\|\|\mathbf{Y}\|$
15	C_{10}	$\|\mathbf{X}\|$ or $\|\mathbf{X}\|\|\mathbf{Y}\|$	$C_{10} \times \|A\|\|\mathbf{X}\|$ or $C_{10} \times \|A\|\|\mathbf{X}\|\|\mathbf{Y}\|$
17	C_{11}	1	$C_{11} \times 1$

In table 1, the values in lines 10 and 15 in the column "Time" have more than one value. The reason for this difference is that searching for an element in a set or adding an element to a set is implemented in at least two ways in mathematical software and programming languages. The faster method is to use a hash table, which has time complexity of $O(1)$. The slower method is the linear search method, which has time complexity of $O(n)$.

To obtain the total time $T(n)$ for $n = |A|$, it is sufficient to sum all the values of the right column of the table 1 together. On the other hand, to evaluate the order of time complexity of an algorithm, it is enough to consider the sentence with the highest degree, that has most effect on the values of $T(n)$. With these explanations and the previous paragraph, function $T(n)$ has two values: $T(n) = C_8|A||\mathbf{X}|^2|R| + \cdots$ and $T(n) = (C_6 + C_7 + C_8)|A||\mathbf{X}||R| + (C_9 + C_{10})|A||\mathbf{X}||\mathbf{Y}| + \cdots$.

In Algorithm 3.3, the number of members of sets $\mathbf{X}$ and $\mathbf{Y}$ changes in the for loops execution, but A and R are constant. Let $|A| = n$. To evaluate the time complexity of the algorithm, an upper bound for R, $\mathbf{X}$ and $\mathbf{Y}$ can be considered. We consider two cases:

First case: *Average–case*
Let $C = 3 * \text{Max}\{C_1, C_2, \ldots, C_{11}\}$ and assume that $0 < |\mathbf{X}|, |\mathbf{Y}|, |R| \leq k \cdot n$, where k is a constant number and $k > 0$ and $n = |A|$.
Hence, in linear search method we get $T(n) \leq 2 \cdot C \cdot n \cdot k^2 \cdot n^2 \cdot k \cdot n + \cdots = 2 \cdot C \cdot k^3 \cdot n^4 + \cdots$. So, we have $\boxed{T(n) \in O(n^4)}$. But in hash table method we get $T(n) \leq 2 \cdot C \cdot n \cdot k \cdot n \cdot k \cdot n + \cdots = 2 \cdot C \cdot k^2 \cdot n^3 + \cdots$. So, we have $\boxed{T(n) \in O(n^3)}$.

Second case: *Worst–case*
Let $C = 3 * \text{Max}\{C_1, C_2, \ldots, C_{11}\}$. Since $\mathbf{X}, \mathbf{Y}, R \subseteq A \times A$ and $|A| = n$, we assume that $0 < |\mathbf{X}|, |\mathbf{Y}|, |R| \leq n^2$. So, we set $|\mathbf{X}|, |\mathbf{Y}|, |R| \leq k \cdot n^2$, where k is a constant number and $0 < k \leq 1$. Hence, in linear search method we get $T(n) \leq 2 \cdot C \cdot n \cdot k^2 \cdot n^4 \cdot k \cdot n^2 + \cdots = 2 \cdot C \cdot k^3 \cdot n^7 + \cdots$. So, we have $\boxed{T(n) \in O(n^7)}$. But, in hash table method we get $T(n) \leq 2 \cdot C \cdot n \cdot k \cdot n^2 \cdot k \cdot n^2 + \cdots = 2 \cdot C \cdot k^2 \cdot n^5 + \cdots$. So, we have $\boxed{T(n) \in O(n^5)}$.

Remark 3.14. According to the above explanations, it seems that the time complexity of Algorithm 3.3 in real problems is between the above two orders, i.e. $O(n^4) \leq T(n) \leq O(n^7)$ in linear search method, and $O(n^3) \leq T(n) \leq O(n^5)$ in hash table method.

Example 3.15. As Example 3.13, let $R = \{(a,b),(b,a),(b,c),(c,d)\}$ and $A = \{a,b,c,d\}$. Since $|A| = 4$, we must calculate R, R^2, R^3 and R^4. The transitive closure of R (i.e. R^*) is obtained using Algorithm 3.2, Algorithm 3.3 or Theorem 3.11 as follows:

(1) $R = \{(a,b),(b,a),(b,c),(c,d)\}$,
(2) $R \cup R^2 = \{(a,a),(a,b),(a,c),(b,a),(b,b),(b,c),(b,d),(c,d)\}$,
(3) $R \cup R^2 \cup R^3 = \{(a,a),(a,b),(a,c),(a,d),(b,a),(b,b),(b,c),(b,d),(c,d)\}$,
(4) $R \cup R^2 \cup R^3 \cup R^4 = \{(a,a),(a,b),(a,c),(a,d),(b,a),(b,b),(b,c),(b,d),(c,d)\}$.

Hence $R^* = \{(a,a),(a,b),(a,c),(a,d),(b,a),(b,b),(b,c),(b,d),(c,d)\}$.

4. Warshall's Algorithm

Based on [6, p. 634] and [5, p. 45], Warshall's algorithm, which is described in 1962 by Stephen Warshall, is an efficient method for computing the transitive closure of a relation. Algorithm 3.1 obtain the transitive closure matrix of a relation on a finite set with n elements using $O(n^4)$ bit operations. However, Warshall's Algorithm (4.1) can compute the matrix of transitive closure using only $O(n^3)$ bit operations. In this section we present this algorithm in two mode. First mode is matrix form that given in subsection 4.1. The necessary preliminaries along with the matrix form of Warshall's algorithm and proofs are also given from [6, pp. 634-637]. Its set-theoretic form is also presented in subsection 4.2 along with the necessary preliminaries.

4.1. The Matrix Form Of Warshall's Algorithm.

The matrix form of Warshall's algorithm is based on the construction of a sequence of zero–one matrices. We denote these matrices by $\mathbf{M}_{\mathbf{W}_0}, \mathbf{M}_{\mathbf{W}_1}, \ldots, \mathbf{M}_{\mathbf{W}_n}$, which is defined as follows.

Definition 4.1. Let R is a relation on a set $A = \{a_1, a_2, \ldots, a_n\}$, where $|A| = n$. *Warshall's matrices* $\mathbf{M}_{\mathbf{W}_i}$ is defined as follows:

$$\begin{cases} \mathbf{M}_{\mathbf{W}_0} = \mathbf{M}_R, \\ \mathbf{M}_{\mathbf{W}_k} = [w_{ij}]_{n\times n}\ ,\ 1 \leq k \leq n\ ,\ w_{ij} = \begin{cases} 1 & (a_i,a_j) \in R^{\infty} \text{ and } V_{a,b} = \{a_1,a_2,\ldots,a_k\}, \\ 0 & \text{o.w.} \end{cases} \end{cases}$$

In other words, $\mathbf{M}_{\mathbf{W}_k} = [w_{ij}]_{n\times n}$, where $w_{ij} = 1$ if there is a path from a_i to a_j such that all the interior vertices of this path are in the set $\{a_1, a_2, \ldots, a_k\}$ and otherwise is 0. The first and last vertices in the path may be outside the set of the first k vertices in the list.

Proposition 4.2 explains the wisdom of Definition 4.1.

Proposition 4.2. *Let R is a relation on a set $A = \{a_1, a_2, \ldots, a_n\}$ where $|A| = n$. Then we have $\mathbf{M}_{\mathbf{W}_n} = \mathbf{M}_{R^*}$.*

Theorem 4.3. *Let $\mathbf{M}_{\mathbf{W}_k} = [w_{ij}^{[k]}]$ be the zero–one matrix so that has a 1 in its (i,j)th position if and only if there is a path from a_i to a_j with interior vertices from the set $\{a_1, a_2, \ldots, a_k\}$. Then $w_{ij}^{[k]} = w_{ij}^{[k-1]} \vee (w_{ik}^{[k-1]} \wedge w_{kj}^{[k-1]})$, whenever i, j, and k are positive integers not exceeding n.*

Theorem 4.3 gives a tool that we can compute the matrices $\mathbf{M}_{\mathbf{W}_k}$, $k = 1, 2, \ldots, n$, efficiently. The Warshall's algorithm, by using Proposition 4.2 and Theorem 4.3, is Algorithm 4.1.

Algorithm 4.1: Matrix form of Warshall's Algorithm

```
Input: M_R: n × n zero–one matrix of a relation R
Output: M_{R*}: n × n zero–one matrix of transitive closure of relation R
W = M_R
for k := 1 to n do
    for i := 1 to n do
        for j := 1 to n do
            w_ij = w_ij ∨ (w_ik ∧ w_kj)
        end for
    end for
end for
return W
```

Based on [6, p. 637], The complexity of Algorithm 4.1 is $O(n^3)$ bit operations.

Example 4.4. As Example 3.13, let $R = \{(a,b),(b,a),(b,c),(c,d)\}$ and $A = \{a,b,c,d\}$. We assume $a_1 = a$, $a_2 = b$, $a_3 = c$, $a_4 = d$. Since $|A| = 4$, we must calculate $\mathbf{M}_{\mathbf{W}_0}$, $\mathbf{M}_{\mathbf{W}_1}$, $\mathbf{M}_{\mathbf{W}_2}$, $\mathbf{M}_{\mathbf{W}_3}$ and $\mathbf{M}_{\mathbf{W}_4}$. The transitive closure of R (i.e. R^*) is obtained using Algorithm 4.1 as follows:

(1) $\mathbf{M}_{\mathbf{W}_0} = \mathbf{M}_R = \begin{bmatrix} 0&1&0&0\\ 1&0&1&0\\ 0&0&0&1\\ 0&0&0&0 \end{bmatrix}$,

(2) $\mathbf{M}_{\mathbf{W}_1} = \begin{bmatrix} 0&1&0&0\\ 1&1&1&0\\ 0&0&0&1\\ 0&0&0&0 \end{bmatrix}$,

(3) $\mathbf{M}_{\mathbf{W}_2} = \begin{bmatrix} 1&1&1&0\\ 1&1&1&0\\ 0&0&0&1\\ 0&0&0&0 \end{bmatrix}$,

(4) $\mathbf{M}_{\mathbf{W}_3} = \begin{bmatrix} 1&1&1&1\\ 1&1&1&1\\ 0&0&0&1\\ 0&0&0&0 \end{bmatrix}$,

(5) $\mathbf{M}_{\mathbf{W}_4} = \begin{bmatrix} 1&1&1&1\\ 1&1&1&1\\ 0&0&0&1\\ 0&0&0&0 \end{bmatrix}$.

Hence $R^* = \{(a,a),\ (a,b),\ (a,c),(a,d),\ (b,a),(b,b),\ (b,c),(b,d),\ (c,d)\}$.

4.2. **Set-Theoretic Form Of Warshall's Algorithm.** In subsection 4.1, Warshall's algorithm was given in matrix form. In this section, we are going to create the Warshall's algorithm in set-theoretic form. The process of doing this is similar to the process done in section 4.1, except that it is done in set-theoretic form instead of matrix form. First, the necessary preliminaries are given and then the corresponding algorithm.

Definition 4.5. Let R is a relation on a set $A = \{a_1, a_2, \ldots, a_n\}$ where $|A| = n$. *Warshall's sets* $\mathbf{W}_k$ is defined as follows:

$$\begin{cases} \mathbf{W}_0 = R, \\ \mathbf{W}_k = \{(a,b) \in R^{\infty} \mid V_{a,b} \subseteq \{a_1, a_2, \ldots, a_k\}\}, \quad \forall k:\ 1 \le k \le n \end{cases}$$

In other words, $(a,b) \in \mathbf{W}_k$ if there is a path from a to b with all the interior vertices of this path in the set $\{a_1, a_2, \ldots, a_k\}$.

Proposition 4.6. *Let R is a relation on a set $A = \{a_1, a_2, \ldots, a_n\}$ where $|A| = n$. Then we have $\mathbf{W}_n = R^*$.*

Proof. By Theorem 3.9 we have $R^* = R^\infty$. On the other hand, by Definition 3.5 we have $(a, b) \in R^\infty$ if and only if there is a path from a to b, with all interior vertices in the set $A = \{a_1, a_2, \ldots, a_n\}$, if and only if $(a, b) \in R^\infty$ and $V_{a,b} = \{a_1, a_2, \ldots, a_n\}$, if and only if $(a, b) \in \mathbf{W}_n$. □

Theorem 4.7. *Let R is a relation on a set $A = \{a_1, a_2, \ldots, a_n\}$ where $|A| = n$. Then we have*

$$\begin{cases} \mathbf{W}_0 = R, \\ \mathbf{W}_{k+1} = \mathbf{W}_k \cup \{(a, b) \in A \times A \big| \{(a, a_{k+1}), (a_{k+1}, b)\} \subseteq \mathbf{W}_k\}, \ \forall k : \ 0 \leq k \leq n \end{cases}$$

Proof. Let $\mathbf{X} = \{(a, b) \in A \times A \big| \{(a, a_{k+1}), (a_{k+1}, b)\} \subseteq \mathbf{W}_k\}$ and k is a positive integer with $0 \leq k \leq n$. It is enough to prove $\mathbf{W}_{k+1} = \mathbf{W}_k \cup \mathbf{X}$.

By Definition 4.5, it is clear that $\mathbf{W}_k \subseteq \mathbf{W}_{k+1}$. Now suppose $(a, b) \in \mathbf{X}$. So (a, a_{k+1}), $(a_{k+1}, b) \in \mathbf{W}_k$. Thus by Definition 4.5, we get $(a, b) \in \mathbf{W}_{k+1}$. Hence we have $\mathbf{X} \subseteq \mathbf{W}_{k+1}$. Therefore, it is concluded that $\mathbf{W}_k \cup \mathbf{X} \subseteq \mathbf{W}_{k+1}$.

Conversely, to prove $\mathbf{W}_{k+1} \subseteq \mathbf{W}_k \cup \mathbf{X}$, suppose $(a, b) \in \mathbf{W}_{k+1}$. From $(a, b) \in \mathbf{W}_{k+1}$, by Definition 4.5 we have $(a, b) \in R^\infty$ and $V_{a,b} \subseteq \{a_1, a_2, \ldots, a_k, a_{k+1}\}$. There are two cases:
Case 1: $a_{k+1} \notin V_{a,b}$.
So, we have $V_{a,b} \subseteq \{a_1, a_2, \ldots, a_k\}$. Therefore, by Definition 4.5 it is concluded that $(a, b) \in \mathbf{W}_k$. This means $(a, b) \in \mathbf{W}_k \cup \mathbf{X}$.

Case 2: $a_{k+1} \in V_{a,b}$.
Since $(a, b) \in R^\infty$, by Definitions 3.5 and 3.1 there is a path from a to b, such as $ax_1 \cdots x_t b$. By Lemma 3.10, this path changes to a path like $ay_1 \cdots y_m b$ in which $m \leq n$ and there are no repeated interior vertices. If a_{k+1} is an interior vertex in this new path, then (a, a_{k+1}), $(a_{k+1}, b) \in \mathbf{W}_k$. So we have $(a, b) \in \mathbf{X} \subseteq \mathbf{W}_k \cup \mathbf{X}$. If a_{k+1} is not an interior vertex in this new path, then $(a, b) \in \mathbf{W}_k \subseteq \mathbf{W}_k \cup \mathbf{X}$. Therefore, we conclude $(a, b) \in \mathbf{W}_k \cup \mathbf{X}$. □

Now, using Proposition 4.6 and Theorem 4.7, the Warshall's algorithm can be given in set-theoretic form.

Algorithm 4.2: Set-theoretic form of Warshall's Algorithm

```
Input: Binary relation R ⊆ A × A, A = {a_1, a_2, ..., a_n}
Output: Transitive closure: R*
W := R
for k ∈ A do
    X := {(a, b) ∈ A × A | {(a, k), (k, b)} ⊆ W}
    W := W ∪ X
end for
return W
```

To evaluate the time complexity of Warshall's Algorithm 4.2, we first rewrite it in terms of basic operations of the sets.

Algorithm 4.3: Set-theoretic form of Warshall's Algorithm (detailed form)

```
1  Input: Binary relation R ⊆ A × A, A = {a_1, a_2, ..., a_n}
2  Output: Transitive closure: R*
3  W := R
4  for k ∈ A do
5  |  X := ∅
6  |  for (a, b) ∈ W do
7  |  |  for (c, d) ∈ W do
8  |  |  |  if b = c = k then
9  |  |  |  |  X := X ∪ {(a, d)}
10 |  |  |  end if
11 |  |  end for
12 |  end for
13 |  W := W ∪ X
14 end for
15 return W
```

In Algorithm 4.2 and Algorithm 4.3, variable $\mathbf{X}$ means the set $\mathbf{X}$ in the proof of Theorem 4.7. It should be noted that in line 5 of Algorithm 4.2 and line 9 of Algorithm 4.3, each member added to the set $\mathbf{X}$ is a new member. The reason for this goes back to details of definition of $\mathbf{X}$ (see the proof of Theorem 4.7).

To analyse the time complexity of Algorithm 4.3, like Algorithm 3.3, we use the RAM model. The details of this model are given in the analysis of Algorithm 3.3. First, we form the table of the time function $T(n)$ of Algorithm 4.3.

TABLE 2. Complexity of Algorithm 4.3

Line	Cost	Time	Total time of the step in algorithm
3	C_1	1	$C_1 \times 1$
4	C_2	$\|A\| + 1$	$C_2 \times (\|A\| + 1)$
5	C_3	1	$C_3 \times n$
6	C_4	$\|\mathbf{W}\| + 1$	$C_4 \times (\|A\|(\|\mathbf{W}\| + 1))$
7	C_5	$\|\mathbf{W}\| + 1$	$C_5 \times (\|A\|\|\mathbf{W}\|(\|\mathbf{W}\| + 1))$
8	C_6	1	$C_6 \times (\|A\|\|\mathbf{W}\|^2)$
9	C_7	1 or $\|\mathbf{X}\|$	$C_7 \times (\|A\|\|\mathbf{W}\|^2)$ or $C_7 \times (\|A\|\|\mathbf{X}\|\|\mathbf{W}\|^2)$
13	C_8	$\|\mathbf{X}\|$ or $\|\mathbf{X}\|\|\mathbf{W}\|$	$C_8 \times (\|A\|\|\mathbf{X}\|)$ or $C_8 \times (\|A\|\|\mathbf{X}\|\|\mathbf{W}\|)$
15	C_9	1	$C_9 \times 1$

Similar to Algorithm 3.3, to evaluate the total time $T(n)$ for $n = |A|$, it is enough to sum all the values of the right column of the table 2 together. On the other hand, to evaluate the order of time complexity of an algorithm, it is enough to consider the sentence with the highest degree, that has most effect on the values of $T(n)$. With these explanations and similar to the analysis of Algorithm 3.3, $T(n)$ has two values: $T(n) = C_7 \times |A||\mathbf{W}|^2 + C_8 \times |A||\mathbf{X}||\mathbf{W}| + \cdots$ and $T(n) = C_7 \times |A||\mathbf{X}||\mathbf{W}|^2 + \cdots$.

In Algorithm 4.3, the number of members of sets $\mathbf{W}$ and $\mathbf{X}$ changes in the for loops execution. Hence, to evaluate the time complexity of the algorithm, an upper bound for R, $\mathbf{W}$ and $\mathbf{X}$ can be considered. We consider two cases:

First case: *Average–case*
Let $C = \text{Max}\{C_1, C_2, \ldots, C_9\}$ and assume that $0 < |\mathbf{W}|, |\mathbf{X}|, |R| \leq k \cdot n$, where k is a constant number and $k > 0$ and $n = |A|$.
Hence, in linear search method we get $T(n) \leq C \cdot n \cdot k \cdot n \cdot k^2 \cdot n^2 + \cdots = C \cdot k^3 \cdot n^4 + \cdots$. So, we have $\boxed{T(n) \in O(n^4)}$. But in hash table method we get $T(n) \leq 2 \cdot C \cdot n \cdot k^2 \cdot n^2 + \cdots = 2 \cdot C \cdot k^2 \cdot n^3 + \cdots$. So, we have $\boxed{T(n) \in O(n^3)}$.

Second case: *Worst–case*
Let $C = \text{Max}\{C_1, C_2, \ldots, C_9\}$. Since $\mathbf{W}, \mathbf{X}, R \subseteq A \times A$ and $|A| = n$, we have $0 < |\mathbf{W}|, |\mathbf{X}|, |R| \leq n^2$. So, we assume $|\mathbf{W}|, |\mathbf{X}|, |R| \leq k \cdot n^2$, where k is a constant number and $0 < k \leq 1$. Hence, in linear search method we get $T(n) \leq C \cdot n \cdot k \cdot n^2 \cdot k^2 \cdot n^4 = C \cdot k^3 \cdot n^7 + \cdots$. So, we have $\boxed{T(n) \in O(n^7)}$. But in hash table method we get $T(n) \leq 2 \cdot C \cdot n \cdot k^2 \cdot n^4 + \cdots = 2 \cdot C \cdot k^2 \cdot n^5 + \cdots$. So, we have $\boxed{T(n) \in O(n^5)}$.

Remark 4.8. According to the above explanations, it seems that the time complexity of Warshall's Algorithm 4.3 in real problems is between the above two orders, i.e. $O(n^4) \leq T(n) \leq O(n^7)$ in linear search method, and$O(n^3) \leq T(n) \leq O(n^5)$ in hash table method.

Considering Remarks 3.14 and 4.8, It can be concluded that the complexity of Warshall's algorithm in matrix form (4.1) is lower than Algorithm 3.1. But the complexity of Warshall's algorithm in set-theoretic form (Algorithm 4.2 or 4.3) is equal to Algorithm 3.2 or 3.3. On the other hand, examining these two algorithms shows that the memory consumption and number of calculations of Algorithm 4.3 are less than Algorithm 3.3, and this is a slight improvement in Algorithm 4.3 compared to Algorithm 3.3.

Example 4.9. As Example 3.13, let $R = \{(a,b),(b,a),(b,c),(c,d)\}$ and $A = \{a,b,c,d\}$. We assume $a_1 = a$, $a_2 = b$, $a_3 = c$, $a_4 = d$. Since $|A| = 4$, we must calculate $\mathbf{W}_0$, $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$ and $\mathbf{W}_4$. The transitive closure of R (i.e. R^*) is obtained using Algorithm 4.2 or Algorithm 4.3 as follows:

(1) $\mathbf{W}_0 = R = \{(a,b),(b,a),(b,c),(c,d)\}$,
(2) $\mathbf{W}_1 = \mathbf{W}_0 \cup \{(x,y) \in A \times A \big| \{(x,a),(a,y)\} \subseteq \mathbf{W}_0\} = \mathbf{W}_0 \cup \{(b,b)\} = \{(a,b),(b,a),(b,b),(b,c),(c,d)\}$,
(3) $\mathbf{W}_2 = \mathbf{W}_1 \cup \{(x,y) \in A \times A \big| \{(x,a),(a,y)\} \subseteq \mathbf{W}_1\} = \mathbf{W}_1 \cup \{(a,a),(a,c)\} = \{(a,a),(a,b),(a,c),(b,a),(b,b),(b,c),(c,d)\}$,
(4) $\mathbf{W}_3 = \mathbf{W}_2 \cup \{(x,y) \in A \times A \big| \{(x,a),(a,y)\} \subseteq \mathbf{W}_2\} = \mathbf{W}_2 \cup \{(a,d),(b,d)\} = \{(a,a),(a,b),(a,c),(a,d),(b,a),(b,b),(b,c),(b,d),(c,d)\}$,
(5) $\mathbf{W}_4 = \mathbf{W}_3 \cup \{(x,y) \in A \times A \big| \{(x,a),(a,y)\} \subseteq \mathbf{W}_3\} = \mathbf{W}_3 \cup \emptyset = \{(a,a),(a,b),(a,c),(a,d),(b,a),(b,b),(b,c),(b,d),(c,d)\}$.

Hence $R^* = \{(a,a),(a,b),(a,c),(a,d),(b,a),(b,b),(b,c),(b,d),(c,d)\}$.

4.3. **Maple codes for algorithms 3.3 and 4.3.** Since Algorithms 3.2, 3.3, 4.2 and 4.3 are new, the Maple codes of these algorithms (Algorithms 3.3 and 4.3) are presented along with their implementation on Example 3.13. You may copy and paste below codes in Maple and execute them.

First, we define two procedures in Maple in Listing 1:

LISTING 1. TransitiveClosure and TransitiveClosurebyWarshall Procedures

```
TransitiveClosure:=proc(R::set,n::integer)::set;
 local i,x,r,T,X,Y;
 X:=R;
 Y:=R;
 for i from 2 to n do
  T:={};
  for x in X do
   for r in R do
    if x[2]=r[1] then
     T:=T union {[x[1],r[2]]};
    end if;
   end do;
  end do;
  X:=T;
  Y:=Y union T;
 end do;
 return Y;
 end proc:
#-----------------------------------------------------------
TransitiveClosurebyWarshall:=proc(R::set,A::set)::set;
 local x,y,k,n,W,X;
 W:=R;
 for k in A do
  X:={};
  for x in W do
   for y in W do
    if x[2]=k and y[1]=k then
     X:=X union {[x[1],y[2]]};
    end if;
   end do;
  end do;
  W:=W union X;
 end do;
 return W;
 end proc:
```

The first procedure is related to Algorithm 3.3 and the second procedure is related to Algorithm 4.3. Now, all we need to do is write Example 3.13 in Maple and call the above procedures on it as follow to achieve the desired result.

```
A:={a,b,c,d};
R:={[a,b],[b,a],[b,c],[c,d]};
TransitiveClosure(R,numelems(A));
TransitiveClosurebyWarshall(R,A);
```

5. Conclusion

Warshall's algorithm has been known for many years as the most optimal algorithm for computing the transitive closure of a relation, but its set-theoretic form was not presented until now.

We considered the Warshall's algorithm, as well as the primitive algorithm that exists for computing the transitive closure. We also presented the set-theoretic form of Warshall's algorithm and evaluated its time complexity order. We described the steps of this algorithms by providing examples. Algorithms are presented in such a way that they can be easily implemented in programming languages or mathematical software (such as Maple). Also, the Maple codes of Algorithms 3.3 and 4.3 along with an example are provided.

Since mathematical software and programming languages support set theory instructions, it is appropriate to research to present algorithms that use matrix form for set-theoretic computations, in set-theoretic form, so that mathematical programs can be more easily implemented on computers. Also, optimization of set-theoretic algorithms can be a research topic in this field.

Acknowledgments Authors are thankful to the reviewers of the paper for providing key suggestions and recommendations to revise and improve the paper.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, fourth edition.* MIT Press, 2022.

[2] E. Nuutila. Efficient transitive closure computation in large digraphs. *Acta Polytechnica Scandinavia: Math. Comput. Eng.*, **74**(1995), 1–124.

[3] S. Pemmaraju and S. Skiena. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica ®*. Cambridge University Press, 2003.

[4] A. Pourhaghani, S. M. Anvariyeh, and B. Davvaz. An algorithm to calculate the members of the relation β in hypergroupoids. *Journal of Algebraic Hyperstructures and Logical Algebras*, **5**(2)(2024), 137–152.

[5] K. H. Rosen. *Handbook of Discrete and Combinatorial Mathematics, Second Edition.* Chapman and Hall/CRC, 2018.

[6] K. H. Rosen. *Discrete Mathematics and Its Applications.* McGraw Hill, 8th edition, 2019.

[7] B. Roy. Transitivité et connexité. *C. R. Acad. Sci. Paris*, **249**(1959), 216–218.

[8] S. S. Skiena. *The Algorithm Design Manual.* Springer International Publishing, 2020.

[9] S. Warshall. A theorem on boolean matrices. *Journal of the ACM*, **9**(1)(1962), 11–12.

[10] E. W. Weisstein. Floyd-warshall algorithm. *Wolfram Research, Inc.*, (2008), 1.