Annales Mathematicae et Informaticae
54 (2021) pp. 17–31
DOI: https://doi.org/10.33039/ami.2021.08.001
URL: https://ami.uni-eszterhazy.hu

Warshall's algorithm—survey and applications

Zoltán Kása

Sapientia Hungarian University of Transylvania, Cluj-Napoca, Romania
Department of Mathematics and Informatics, Târgu Mureş
kasa@ms.sapientia.ro

Submitted: December 23, 2020
Accepted: August 3, 2021
Published online: August 13, 2021

Abstract

This survey presents the well-known Warshall's algorithm, a generalization and some interesting applications: transitive closure of relations, distances between vertices in graphs, number of paths in acyclic digraphs, all paths in digraphs, scattered complexity for rainbow words, special walks in finite automata.

Keywords: Warshall's algorithm, Floyd–Warshall algorithm, paths in graphs, scattered subword complexity, finite automata

AMS Subject Classification: 05C85, 68W05, 68R10, 68R14

1. Introduction

Warshall's algorithm [14] with its generalization [11] is widely used in graph theory [1, 3–5, 8–10, 12] and in various fields of sciences (e.g. fuzzy [13] and quantum [6] theory, Kleene algebra [7]). In the following, we present the algorithm, its generalization, and the collected applications related to graphs, to be easily accessible together here.

Let R be a binary relation on the set $S = \{s_1, s_2, \ldots, s_n\}$, we write $s_i R s_j$ if s_i is in relation with s_j. The relation R can be represented by the so called *relation matrix*, which is

$$A = (a_{ij})_{\substack{i=\overline{1,n} \\ j=\overline{1,n}}}, \quad \text{where} \quad a_{ij} = \begin{cases} 1, & \text{if } s_i R s_j, \\ 0, & \text{otherwise.} \end{cases}$$

The transitive closure of the relation R is the binary relation R^* defined as: $s_i R^* s_j$ if and only if there exists $s_{p_1}, s_{p_2}, \ldots, s_{p_r}, r \geq 2$ such that $s_i = s_{p_1}$, $s_{p_1} R s_{p_2}$, $s_{p_2} R s_{p_3}, \ldots, s_{p_{r-1}} R s_{p_r}$, $s_{p_r} = s_j$. The relation matrix of R^* is $A^* = (a^*_{ij})$.

Let us define the following two operations: i) if $a, b \in \{0, 1\}$ then $a + b = 0$ for $a = 0$, $b = 0$, and $a + b = 1$ otherwise; ii) $a \cdot b = 1$ for $a = 1, b = 1$, and $a \cdot b = 0$ otherwise. In this case

$$A^* = A + A^2 + \cdots + A^n.$$

The transitive closure of a relation can be computed easily by the Warshall's algorithm [2, 14]:

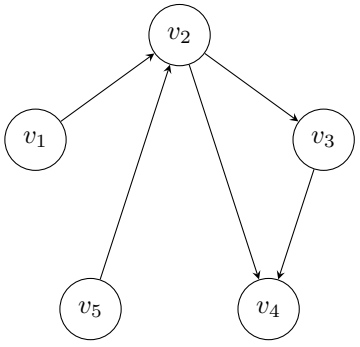
WARSHALL(A, n)
Input: the relation matrix A; the number of elements n
Output: $W = A^*$

```
1  W ← A
2  for k ← 1 to n
3      do for i ← 1 to n
4              do for j ← 1 to n
5                      do if w_ik = 1 and w_kj = 1
6                              then w_ij ← 1
7  return W
```

Listing 1. Warshall's algorithm.

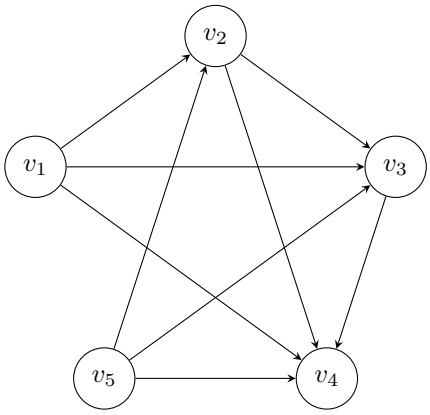
The complexity of this algorithm is $\Theta(n^3)$.

A binary relation can be represented by a directed graph (i.e. digraph) too. The relation matrix is equal to the adjacency matrix of the corresponding graph. See Fig. 1 for an example. Fig. 2 represents the graph of the corresponding transitive closure relation.



$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Figure 1. A binary relation represented by a graph with the corresponding adjacency matrix.



$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 2. The transitive closure of the relation in Fig. 1.

2. Generalization of Warshall's algorithm

Lines 5 and 6 in the Warshall's algorithm presented in Listing 1 can be expressed as

$$w_{ij} \leftarrow w_{ij} + w_{ik} \cdot w_{kj}$$

using the operations defined above. If instead of the operations $+$ and $\cdot$ we use two operations $\oplus$ and $\odot$ from a semiring, a generalized Warshall's algorithm results [11]:

GENERALIZED-WARSHALL(A, n)
Input: the relation matrix A; the number of elements n
Output: $W = A^*$

```
1  W ← A
2  for k ← 1 to n
3      do for i ← 1 to n
4              do for j ← 1 to n
5                      do w_ij ← w_ij ⊕ (w_ik ⊙ w_kj)
6  return W
```

Listing 2. The generalized Warshall's algorithm.

The complexity of this algorithm is also $\Theta(n^3)$. This generalization leads us to a number of interesting applications.

3. Applications

3.1. Distances between vertices. Floyd–Warshall algorithm

Given a (di)graph with positive or negative edge weights (but with no negative cycles) and its modified adjacency matrix $D_0 = (d_{ij}^0)$, we can obtain the distance matrix $D = (d_{ij})$ in which d_{ij} represents the distance between vertices v_i and v_j. The distance between vertices v_i and v_j is the length of the shortest path between them. The modified adjacency matrix $D_0 = (d_{ij}^0)$ is the following:

$$d_{ij}^0 = \begin{cases} 0, & \text{if } i = j, \\ \infty, & \text{if there is no edge from vertex } v_i \text{ to vertex } v_j, i \neq j, \\ w_{ij}, & \text{the weight of the edge from } v_i \text{ to } v_j, i \neq j. \end{cases}$$

Choosing for $\oplus$ the min operation (minimum of two real numbers), and for $\odot$ the real addition (+), we obtain the well-known Floyd–Warshall algorithm as a special case of the generalized Warshall's algorithm [5, 11, 12] :

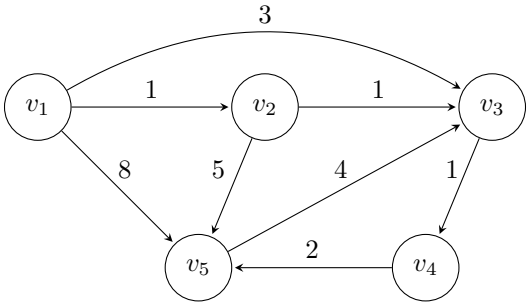
```
FLOYD-WARSHALL(D_0, n)
Input: the adjacency matrix D_0; the number of elements n
Output: the distance matrix D
1  D ← D_0
2  for k ← 1 to n
3      do for i ← 1 to n
4             do for j ← 1 to n
5                    do d_ij ← min{d_ij, d_ik + d_kj}
6  return D
```

Listing 3. Floyd-Warshall algorithm.

An example is presented in Fig. 3. The shortest paths can also be easily obtained by storing the previous vertex v_k on the path, in line 5 of Listing 3. In the case of acyclic digraphs, the algorithm can be easily modified to obtain the longest distances between vertices and consequently the longest paths.

3.2. Number of paths in acyclic digraphs

Here, by path we understand a directed path. In an acyclic digraph the following algorithm counts the number of paths between vertices [4, 9]. The operation $\oplus$, $\odot$ are the classical add and multiply operations for real numbers and let w_{ij} denote the number of paths from vertex v_i to vertex v_j.



$$D_0 = \begin{pmatrix} 0 & 1 & 3 & \infty & 8 \\ \infty & 0 & 1 & \infty & 5 \\ \infty & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & 0 & 2 \\ \infty & \infty & 4 & \infty & 0 \end{pmatrix} \qquad D = \begin{pmatrix} 0 & 1 & 2 & 3 & 5 \\ \infty & 0 & 1 & 2 & 4 \\ \infty & \infty & 0 & 1 & 3 \\ \infty & \infty & 6 & 0 & 2 \\ \infty & \infty & 4 & 5 & 0 \end{pmatrix}$$

Figure 3. A weighted digraph with the corresponding matrices.

WARSHALL-PATHS(A, n)
Input: the adjacency matrix A; the number of elements n
Output: W with number of paths between vertices

```
1  W ← A
2  for k ← 1 to n
3      do for i ← 1 to n
4             do for j ← 1 to n
5                    do w_ij ← w_ij + w_ik · w_kj
6  return W
```

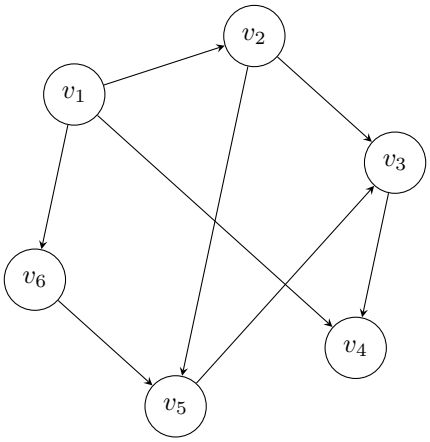
Listing 4. Finding the number of paths between vertices.

The proof is omitted, as it is very similar to the one given in [2] for the original Warshall's algorithm.

An example can be seen in Fig. 4. For example between vertices v_1 and v_3 there are 3 paths: (v_1, v_2, v_3); (v_1, v_2, v_5, v_3) and (v_1, v_6, v_5, v_3).

For the case when the arcs of the graph are colored, we may be interested in the number of monochromatic paths. The generalized algorithm can also be used for monochromatic subgraphs. The following novel algorithm (first described here) solves the problem for all colors at once.

In the adjacency (color) matrix, a_{ij} is equal to the code of the color of the arc (v_i, v_j), and is equal to 0, if there is no arc from v_i to v_j. In the three-dimensional result matrix W the element w_{ijp} represents the number of the paths from v_i to v_j with p-colored arcs each.



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \qquad W = \begin{pmatrix} 0 & 1 & 3 & 4 & 2 & 1 \\ 0 & 0 & 2 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

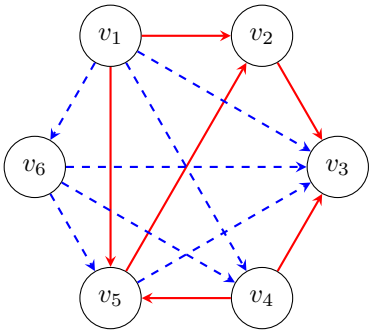
Figure 4. An acyclic digraph and the corresponding matrices.

WARSHALL-MONOCHROMATIC-PATHS(A, n, c)
Input: adjacency color matrix A; number of elements n; number of colors c
Output: matrix W with number of monochromatic paths between vertices

```
 1  Set all elements of W equal to 0
 2  for i ← 1 to n
 3       do for j ← 1 to n
 4            do if a_ij ≠ 0
 5                 do w_{ij a_ij} ← 1
 6  for p ← 1 to c
 7       do for k ← 1 to n
 8            do for i ← 1 to n
 9                 do for j ← 1 to n
10                      do w_ijp ← w_ijp + w_ikp · w_kjp
11  return W
```

Listing 5. Finding the number of monochromatic paths between vertices.

The sides for $p = 1, \ldots, c$ of the three-dimensional matrix W contain the result for the different colors (see Fig. 5).



No. of red paths:

$$W(*,*,1) = \begin{pmatrix} 0 & 2 & 2 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

No. of blue paths:

$$W(*,*,2) = \begin{pmatrix} 0 & 0 & 3 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 1 & 0 \end{pmatrix}$$

Figure 5. An example of a colored digraph with the two sides of the solution matrix.

3.3. All paths in digraphs

The Warshall's algorithm combined with the Latin square method can be used to obtain all paths in a (not necessarily acyclic) digraph [9]. A path will be denoted by a string formed by its vertices in their natural order in the path.

Let us consider a matrix $\mathcal{A}$ with the elements A_{ij} which are a set of strings. Initially, the elements of this matrix are defined as:

$$A_{ij} = \begin{cases} \{v_i v_j\}, & \text{if } \exists \text{ an arc from } v_i \text{ to } v_j,\ i \neq j, \\ \emptyset, & \text{otherwise,} \end{cases} \qquad \text{for } i,j = 1,2,\ldots,n. \tag{3.1}$$

If $\mathcal{A}$ and $\mathcal{B}$ are sets of strings, $\mathcal{AB}$ will be formed by the set of concatenation of each string from $\mathcal{A}$ with each string from $\mathcal{B}$, if they have no common letters:

$$\mathcal{AB} = \big\{ab \,\big|\, a \in \mathcal{A}, b \in \mathcal{B},\ \text{if } a \text{ and } b \text{ have no common letters}\big\}. \tag{3.2}$$

If $s = s_1 s_2 \cdots s_p$ is a string, let us denote by $'s$ the string obtained from s by eliminating the first character: $'s = s_2 s_3 \cdots s_p$. Let us denote by $'A_{ij}$ the set A_{ij} in which we eliminate from each element the first character. In this case $'\mathcal{A}$ is a matrix with elements $'A_{ij}$.

Operations are: set union and set product defined as before.

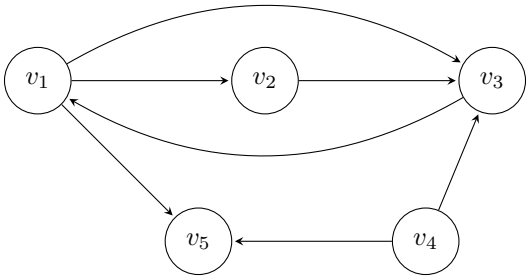
Starting with the matrix $\mathcal{A}$ defined as before, the algorithm to obtain all paths is the following, in which W_{ij} represents the set of paths from vertex v_i to v_j.

WARSHALL-LATIN($\mathcal{A}, n$)
Input: the adjacency matrix $\mathcal{A}$ defined in (3.1); the number of elements n
Output: $\mathcal{W}$ matrix of the paths between vertices

```
1  W ← A
2  for k ← 1 to n
3      do for i ← 1 to n
4              do for j ← 1 to n
5                      do if W_ik ≠ ∅ and W_kj ≠ ∅
6                              then W_ij ← W_ij ∪ W_ik 'W_kj
7  return W
```

Listing 6. Algorithm for finding all paths in digraphs.

An example is presented in Fig. 6: here, for example, between vertices v_1 and v_3 there are two paths: v_1v_3 and $v_1v_2v_3$.



$$\mathcal{A} = \begin{pmatrix} \emptyset & \{v_1v_2\} & \{v_1v_3\} & \emptyset & \{v_1v_5\} \\ \emptyset & \emptyset & \{v_2v_3\} & \emptyset & \emptyset \\ \{v_3v_1\} & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{v_4v_3\} & \emptyset & \{v_4v_5\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

$$\mathcal{W} = \begin{pmatrix} \emptyset & \{v_1v_2\} & \{v_1v_3,\ v_1v_2v_3\} & \emptyset & \{v_1v_5\} \\ \{v_2v_3v_1\} & \emptyset & \{v_2v_3\} & \emptyset & \{v_2v_3v_1v_5\} \\ \{v_3v_1\} & \{v_3v_1v_2\} & \emptyset & \emptyset & \{v_3v_1v_5\} \\ \{v_4v_3v_1\} & \{v_4v_3v_1v_2\} & \{v_4v_3\} & \emptyset & \{v_4v_5\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

Figure 6. An example of digraph for all paths problem with the corresponding matrices.

Although the algorithm is not polynomial (due to the $W_{ik}\,'W_{kj}$ multiplication), the method can be used well in practice for many cases.

3.4. Scattered complexity for rainbow words

The application described in this subsection can be found in [9]. Let Σ be an alphabet, Σ^n the set of all length-n words over Σ, Σ^* the set of all finite word over Σ.

Definition 3.1. Let n and s be positive integers, $M \subseteq \{1, 2, \ldots, n-1\}$ and $u = x_1 x_2 \ldots x_n \in \Sigma^n$. An M**-subword** of length s of u is defined as $v = x_{i_1} x_{i_2} \ldots x_{i_s}$ where

$$\begin{aligned} &i_1 \geq 1, \\ &i_{j+1} - i_j \in M \text{ for } j = 1, 2, \ldots, s-1, \\ &i_s \leq n. \end{aligned}$$

Definition 3.2. The number of M-subwords of a word u for a given set M is the scattered subword complexity, simply M-complexity.

Examples. The word *abcd* has 11 $\{1,3\}$-subwords: *a*, *ab*, *abc*, *abcd*, *ad*, *b*, *bc*, *bcd*, *c*, *cd*, *d*. The $\{2, 3\,4, 5\}$-subwords of the word *abcdef* are the following: *a*, *ac*, *ad*, *ae*, *af*, *ace*, *acf*, *adf*, *b*, *bd*, *be*, *bf*, *bdf*, *c*, *ce*, *cf*, *d*, *df*, *e*, *f*.

Words with different letters are called *rainbow words*. The M-complexity of a length-n rainbow word does not depend on what letters it contains, and is denoted by $K(n, M)$.

To compute the M-complexity of a rainbow word of length n we will use graph theoretical results. Let us consider the rainbow word $a_1 a_2 \ldots a_n$ and the corresponding digraph $G = (V, E)$, with

$$\begin{aligned} V &= \{a_1, a_2, \ldots, a_n\}, \\ E &= \{(a_i, a_j) \mid j - i \in M,\ i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, n\}. \end{aligned}$$

For $n = 6, M = \{2, 3, 4, 5\}$ see Fig. 7.

The adjacency matrix $A = (a_{ij})_{\substack{i=\overline{1,n} \\ j=\overline{1,n}}}$ of the graph is defined by:

$$a_{ij} = \begin{cases} 1, & \text{if } j - i \in M, \\ 0, & \text{otherwise,} \end{cases} \quad \text{for} \quad i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, n.$$

Because the graph has no directed cycles, the element in row i and column j in A^k (where $A^k = A^{k-1}A$, with $A^1 = A$) will represent the number of length-k directed paths from a_i to a_j. If I is the identity matrix (with elements equal to 1 only on the first diagonal, and 0 otherwise), let us define the matrix $R = (r_{ij})$:

$$R = I + A + A^2 + \cdots + A^k, \quad \text{where} \quad k < n,\ A^{k+1} = O \quad \text{(the null matrix)}.$$

The M-complexity of a rainbow word is then

$$K(n, M) = \sum_{i=1}^{n} \sum_{j=1}^{n} r_{ij}.$$

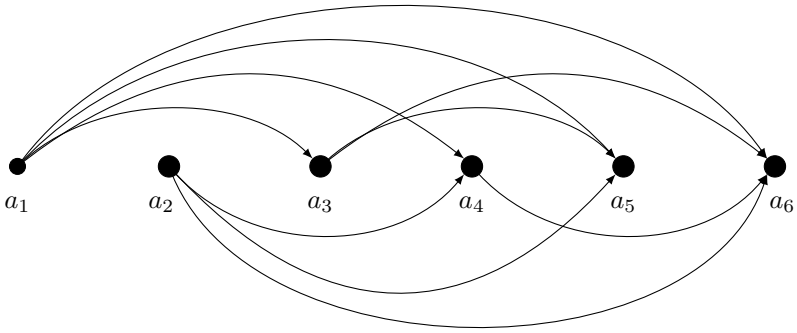


Figure 7. Graph for $\{2,3,4,5\}$-subwords of the rainbow word of length 6.

Matrix R can be better computed using the WARSHALL-PATHS algorithm described in Listing 4, which gives a matrix W, and therefore $R = I + W$.

For example, let us consider the graph in Fig. 7. [9] The corresponding adjacency matrix is:

$$A = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

After applying the WARSHALL-PATHS algorithm:

$$W = \begin{pmatrix} 0 & 0 & 1 & 1 & 2 & 3 \\ 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad R = \begin{pmatrix} 1 & 0 & 1 & 1 & 2 & 3 \\ 0 & 1 & 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

and then $K\big(6, \{2,3,4,5\}\big) = 20$, the sum of elements in R.

Remark 3.3. For this case the WARSHALL-PATHS algorithm can be slightly modified: because of the specific form of the graph the lines 2 and 4 can also be written in the following form:

2 ***for*** $k \leftarrow 2$ ***to*** $n-1$
4 ***do for*** $j \leftarrow i+1$ ***to*** n □

Using the WARSHALL-LATIN algorithm (Listing 6) we can obtain all nontrivial (with length at least 2) M-subwords of a given length-n rainbow word $a_1 a_2 \cdots a_n$.

Let us consider a matrix $\mathcal{A}$ with the elements A_{ij} which form a set of strings. Initially this matrix is defined as:

$$A_{ij} = \begin{cases} \{a_i a_j\}, & \text{if } j-i \in M, \\ \emptyset, & \text{otherwise,} \end{cases} \quad \text{for} \quad i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, n.$$

The set of nontrivial M-subwords is $\bigcup_{i,j\in\{1,2,\ldots,n\}} W_{ij}$.

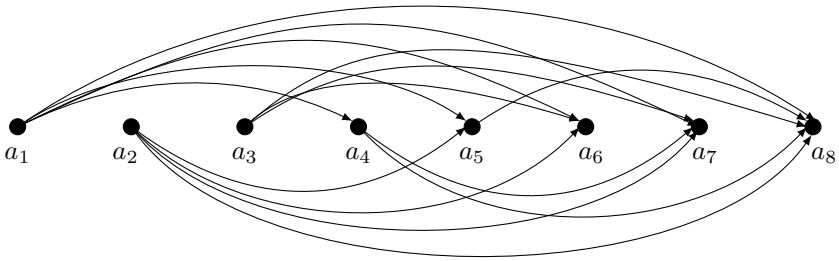


Figure 8. Graph for $\{3, 4, 5, 6, 7\}$-subwords of the rainbow word of length 8.

For $n = 8$, $M = \{3, 4, 5, 6, 7\}$ (see Fig. 8) the initial matrix is:

$$\begin{pmatrix} \emptyset & \emptyset & \emptyset & \{a_1a_4\} & \{a_1a_5\} & \{a_1a_6\} & \{a_1a_7\} & \{a_1a_8\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{a_2a_5\} & \{a_2a_6\} & \{a_2a_7\} & \{a_2a_8\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{a_3a_6\} & \{a_3a_7\} & \{a_3a_8\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{a_4a_7\} & \{a_4a_8\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{a_5a_8\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

The result of the algorithm in this case is:

$$\begin{pmatrix} \emptyset & \emptyset & \emptyset & \{a_1a_4\} & \{a_1a_5\} & \{a_1a_6\} & \{a_1a_7, a_1a_4a_7\} & \{a_1a_8, a_1a_4a_8, a_1a_5a_8\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{a_2a_5\} & \{a_2a_6\} & \{a_2a_7\} & \{a_2a_8, a_2a_5a_8\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{a_3a_6\} & \{a_3a_7\} & \{a_3a_8\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{a_4a_7\} & \{a_4a_8\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{a_5a_8\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}.$$

3.5. Special walks in finite automata

Let us consider a finite automaton $A = (Q, \Sigma, \delta, \{q_1\}, F)$, where Q is a finite set of states, Σ the input alphabet, $\delta\colon Q \times \Sigma \to Q$ the transition function, q_1 the initial state, F the set of final states. In the following, we omit to mark the initial and the final states. The transition function can also be generalized to words: $\delta(q, wa) = \delta(\delta(q, w), a)$, where $q \in Q, a \in \Sigma, w \in \Sigma^*$. A sequence of the form

$$q_1, a_1, q_2, a_2, \ldots, a_{n-1}, q_n, \quad n \geq 2,$$

where

$$\delta(q_1, a_1) = q_2, \delta(q_2, a_2) = q_3, \ldots, \delta(q_{n-1}, a_{n-1}) = q_n$$

is a walk in the automata labelled by the word $a_1 a_2 \ldots a_{n-1}$. This also can be written as:

$$q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-2}} q_{n-1} \xrightarrow{a_{n-1}} q_n,$$

or shortly: $q_1 \xrightarrow{a_1 a_2 \ldots a_{n-1}} q_n$.

We are interested in finding walks with special labels: one-letter power words (power of a single letter) and rainbow words (containing only dissimilar letters).

3.5.1. Walks labeled with one-letter power words

For each pair p, q of states we search for the letters a for which there exists a natural $k \geq 1$ such that we have the transition $\delta(p, a^k) = q$ (see [11]). Let us denote these sets by:

$$W_{ij} = \{a \in \Sigma \mid \exists k \geq 1, \delta(q_i, a^k) = q_j\},$$

where a^k is a length-k one-letter power word.

Here the elements A_{ij} of the adjacency matrix $\mathcal{A}$ initially are defined as:

$$A_{ij} = \{a \mid \delta(q_i, a) = q_j\}, \quad \text{for} \quad i, j = 1, 2, \ldots, n.$$

Instead of $\oplus$ we use here set union ($\cup$) and instead of $\odot$ set intersection ($\cap$).

WARSHALL-AUTOMATA-1$(\mathcal{A}, n)$
Input: the adjacency matrix $\mathcal{A}$; the number of states n
Output: the matrix $\mathcal{W}$ with sets of letters for one letter power words

```
1  W ← A
2  for k ← 1 to n
3      do for i ← 1 to n
4             do for j ← 1 to n
5                    do W_ij ← W_ij ∪ (W_ik ∩ W_kj)
6  return W
```

Listing 7. Finding walks labeled by one-letter power words.

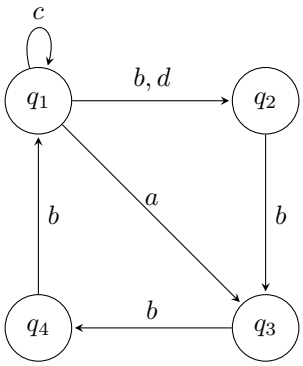


Figure 9. An example of a finite automaton without indicating the initial and final states.

The transition table of the finite automaton in Fig. 9 is:

δ	a	b	c	d
q_1	q_3	q_2	q_1	q_2
q_2	$\emptyset$	q_3	$\emptyset$	$\emptyset$
q_3	$\emptyset$	q_4	$\emptyset$	$\emptyset$
q_4	$\emptyset$	q_1	$\emptyset$	$\emptyset$

Matrices for the graph in Fig. 9 are the following:

$$\mathcal{A} = \begin{pmatrix} \{c\} & \{b,d\} & \{a\} & \emptyset \\ \emptyset & \emptyset & \{b\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \{b\} \\ \{b\} & \emptyset & \emptyset & \emptyset \end{pmatrix}, \quad \mathcal{W} = \begin{pmatrix} \{b,c\} & \{b,d\} & \{a,b\} & \{b\} \\ \{b\} & \{b\} & \{b\} & \{b\} \\ \{b\} & \{b\} & \{b\} & \{b\} \\ \{b\} & \{b\} & \{b\} & \{b\} \end{pmatrix}.$$

For example $\delta(q_2, bb) = q_4$, $\delta(q_2, bbb) = q_1$, $\delta(q_2, bbbb) = q_2$, $\delta(q_1, c^k) = q_1$ for $k \geq 1$.

3.5.2. Walks labeled with rainbow words

To find walks with rainbow labels, we can use the a variant of the Warshall-Latin algorithm (Listing 6), where instead of string of vertices $v_1 v_2 \cdots v_k$ we use the corresponding string of labels of the edges (v_1, v_2), …, (v_{k-1}, v_k).

Here the elements A_{ij} of the adjacency matrix $\mathcal{A}$ are initially defined as:

$$A_{ij} = \{a \mid \delta(q_i, a) = q_j\}, \quad \text{for} \quad i, j = 1, 2, \ldots, n.$$

The concatenation $W_{ik}\, W_{kj}$ in the following algorithm is defined as in the formula (3.2). Each element of W_{ik} is concatenated with each element of W_{kj} only if these elements (which are strings) have no common letters. If a string appears more than once during concatenation, only one copy is retained. The following algorithm is a new one.

```
WARSHALL-AUTOMATA-2(A, n)
Input: the adjacency matrix A; the number of states n
Output: the matrix W of the rainbow words between vertices
1  W ← A
2  for k ← 1 to n
3      do for i ← 1 to n
4             do for j ← 1 to n
5                    do if W_ik ≠ ∅ and W_kj ≠ ∅
6                          then W_ij ← W_ij ∪ W_ik W_kj
7  return W
```

Listing 8. Finding walks labeled by rainbow words.

For the automaton in Fig. 9 the above algorithm uses the matrix $\mathcal{A}$:

$$\mathcal{A} = \begin{pmatrix} \{c\} & \{b,d\} & \{a\} & \emptyset \\ \emptyset & \emptyset & \{b\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \{b\} \\ \{b\} & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

and gives the following result:

$$\mathcal{W} = \begin{pmatrix} \{c\} & \{b,d,cb,cd\} & \{a,ca,db,cdb\} & \{ab,cab\} \\ \emptyset & \emptyset & \{b\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \{b\} \\ \{b,bc\} & \{bd,bcd\} & \{ba,bca\} & \emptyset \end{pmatrix}.$$

Conclusions

In 1962 S. Warshall published the algorithm later named after him for computing the transitive closure of a binary relation [14]. R. W. Floyd reported the application of this in the same year to determine the shortest paths in weighted graphs [5]. P. Robert and J. Ferland in their 1968 article [11] gave an interesting generalization that led to the applications discussed in this article [5, 9, 11, 12]. Two algorithms, WARSHALŁ-MONOCHROMATICS-PATHS and WARSHALL-AUTOMATA-2, are new applications firstly described here.

It is amazing how diverse the applications are. And there can be more!

Acknowledgements. The author thanks the anonymous reviewers for their attentive and thorough work in improving the paper with their helpful remarks.

References

[1] A. Ainia, A. Salehipour: *Speeding up the Floyd–Warshall algorithm for the cycled shortest path problem*, Applied Mathematics Letters 25.1 (2012), pp. 1–5,
doi: https://doi.org/10.1016/j.aml.2011.06.008.

[2] S. Baase: *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, 1983, 1988.

[3] R. Berghammer: *A Functional, Successor List Based Version of Warshall's Algorithm with Applications. Relational and Algebraic Methods in Computer Science. RAMICS, 2011. Lecture Notes in Computer Science, vol 6663*, in: Relational and Algebraic Methods in Computer Science,
doi: https://doi.org/0.1007/978-3-642-21070-9_10.

[4] C. Elzinga, H. Wang: *Kernels for acyclic digraphs*, Pattern Recognition Letters 33.16 (2013), pp. 2239–2244,
doi: https://doi.org/10.1016/j.patrec.2012.07.017.

[5] R. W. Floyd: *Algorithm 97: Shortest Path*, Communications of the ACM 5.6 (1962), p. 345,
doi: https://doi.org/10.1145/367766.368168.

[6] A. S. Gupta, A. Pathak: *Quantum Floyd-Warshall algorithm*, arXiv:quant-ph/0502144.

[7] P. Höfner, B. Möller: *Dijkstra, Floyd and Warshall meet Kleene*, Formal Aspect of Computing 24 (2012), pp. 459–476,
doi: https://doi.org/10.1007/s00165-012-0245-4.

[8] S. Hougardy: *The Floyd–Warshall algorithm on graphs with negative cycles*, Information Processing Letters 110, pp. 279–281,
doi: https://doi.org/10.1016/j.ipl.2010.02.001.

[9] Z. Kása: *On scattered subword complexity*, Acta Univ. Sapientiae Informatica 3.1 (2011), pp. 127–136,
url: acta.sapientia.ro/acta-info/C3-1/info31-6.pdf.

[10] A. Ojo, N. Ma, I. Woungang: *Modified Floyd-Warshall algorithm for equal cost multipath in software-defined data center. 2015 IEEE International Conference on Communication Workshop (ICCW), London*, in: pp. 346–351,
doi: https://doi.org/10.1109/ICCW.2015.7247203.

[11] P. Robert, J. Ferland: *Généralisation de l'algorithme de Warshall*, Revue Française d'Informatique et de Recherche Opérationnelle 2.7 (1968), pp. 71–85,
url: www.numdam.org/item/?id=M2AN_1968__2_1_71_0.

[12] Z. A. Vattai: *Floyd-Warshall again*,
url: www.ekt.bme.hu/Cikkek/54-Vattai_Floyd-Warshall_Again.pdf.

[13] Q. Wang, D. Zhang: *A simple and direct algorithm for computing transitive closure of fuzzy matrix*, Journal of Xi'an University of Technology 3 (2006),
url: en.cnki.com.cn/Article_en/CJFDTOTAL-XALD200603011.htm.

[14] S. Warshall: *A theorem on boolean matrices*, Journal of the ACM 9.1 (1962), pp. 11–12,
doi: https://doi.org/10.1145/321105.321107.