© Algebra and Discrete Mathematics
Volume **42** (2026). Number 1, pp. 93–118
DOI:10.12958/adm2372

RESEARCH ARTICLE

Compression of defining pair for deterministic and strongly deterministic graphs

Oleksii S. Senchenko and Mykola I. Prytula

Communicated by A. Zhuchok

Abstract. For the representation of deterministic and strongly deterministic graphs by defining pairs of (multi)sets of words, we propose a class of equivalent transformations that preserve the graph represented by a correct pair. Based on these elementary transformations, we develop a compression algorithm that transforms an initial correct pair into a canonical defining pair for the same graph. The compression algorithm also provides a method for determining whether a given correct pair is a defining pair for a prescribed deterministic graph. Finally, we outline several possible applications of pair compression.

Introduction

Graphs with vertex labeling are actively used as a model of many software objects, information processes, digital network topologies, etc. In particular, deterministic graphs and their variations, also known as graphs with distance-2 [1] and injective [2] vertex coloring are widely used in various applied tasks, in particular for planning the placement of structural elements and channels in wireless networks [3]. Such planning

This work was partially supported by a grant from the Simons Foundation (Award 1160640, Presidential Discretionary-Ukraine Support Grants, Oleksii Senchenko, Mykola Prytula). We thank Prof. Anatolii V. Zhuchok and the anonymous referee for their suggestions on how to restructure this paper for a clearer presentation style.

2020 Mathematics Subject Classification: *68R10, 05C62, 05C78.*

Key words and phrases: *compression, deterministic graph, presentation, defining pair, minimization, recognizing a correct pair.*

ensures the efficiency of networks in a number of areas, including the construction of minimum coverage, i.e., determining the optimal number of towers and their placement, routing optimization, interference control to reduce radio interference, load balancing, energy consumption optimization, etc. In the context of practical applications of such graphs, an important task is to study the structure and properties of individual subclasses of graphs with vertex labeling. The structure of graphs with labeled vertices, in particular deterministic graphs, is often studied experimentally using mobile agents [4–7], which are placed in the graph, then move along the edges of the graph and report local information to the operator-researcher, based on which the researcher determines the structure of the graph under study. Deterministic graphs, in which all vertices in the neighborhood of each vertex have pairwise distinct labels, have a natural connection to languages in the alphabet of vertex labels: if the vertex from which the agent starts moving is known, the trajectory of this movement along the graph is uniquely determined by the sequence of visited labels. This allows us to consider such graphs in close connection with automata theory and apply linguistic methods to describe them.

In such research, traditional methods of graph specification based on matrices, lists, and dictionaries do not fully cover the possibilities of information processing. In [8], a linguistic presentation of deterministic graphs was proposed using two special sets of words, these two sets being referred to as a "pair". This presentation of a graph by sets of words allows the agent to process data in symbolic form, which can simplify algorithmic procedures. In particular, symbolic encoding of vertices and edges facilitates the search, comparison, and verification of routes, since the agent's transitions can be described as simple operations on strings. This creates an intuitive and compact mechanism for describing agent behavior that does not require complex structures such as adjacency matrices or dictionaries. As a result, the mobile agent can more effectively use text instructions as a standard interface for interacting with the graph, making the approach universal and suitable for a wide range of computer science tasks.

This article clarifies some aspects of the proposed presentation and offers a solution to the problem of characterizing the linguistic presentation of deterministic graphs. The structure of the article is as follows: Section 1 presents the basic definitions and concepts related to deterministic graphs, Section 2 outlines the basic concepts and algorithms related to the linguistic presentation of deterministic graphs by the so-

called defining pair, Section 3 presents equivalent transformations of a pair that simplify its numerical characteristics – the number and length of words in its components. Based on these transformations, Section 4 develops the compression algorithm that transforms a given correct pair (i.e., one that is a presentation for some deterministic graph) into the so-called canonical defining pair; this algorithm is the basis for solving the pair characterization problem under consideration.

1. Main definitions

In this article, we consider undirected, finite, non-empty, connected, simple graphs $G = (V, E)$, where V is the set of vertices of the graph, E is the set of its edges. A graph consisting of a single vertex is called *trivial*. A *walk* (of *length* k) in a graph G is a non-empty alternating sequence $p = v_0e_0v_1e_1 \dots e_{k-1}v_k$ of vertices and edges in G such that $e_i = \{v_i, v_{i+1}\}$ for all $i < k$. A *path* is a walk in which all vertices (and therefore also all edges) are distinct. We denote a walk (or a path) by the sequence of its vertices: $p = v_0v_1 \dots v_k$. The length of the walk (or the path) p is denoted by $|p|$. Note that k is allowed to be zero, in which case the walk consists of a single vertex. If $v_0 = v_k$, the walk was *closed*. A *cycle* is a closed walk in which the first and last vertices are equal, and all other vertices (and therefore also all edges) are distinct. The *open neighborhood* of a vertex $v \in V$ is the set $N(v) = \{u|\{u, v\} \in E\}$, and its *closed neighborhood* is the set $N[v] = N(v) \bigcup \{v\}$. The cardinality of $|N(v)|$ is called the *degree* of v, a vertex of degree 1 is called a *leaf* or a *leaf vertex*.

A *deterministic graph* (or a *D-graph*) is a quintuple $\tilde{G} = (V, E, \Lambda, \xi, v_0)$, where $G = (V, E)$ is a graph, $\Lambda = \{x_1, \dots, x_p\}$ is a finite alphabet, $v_0 \in V$ is an arbitrary *basepoint* vertex, and $\xi : V \to \Lambda$ is a *deterministic* (*locally-bijective*) map, i.e., ξ is bijective on the open neighborhood $N(v)$ of each vertex $v \in V$: $\forall v_1, v_2 \in N(v)(v_1 \neq v_2 \to \xi(v_1) \neq \xi(v_2))$. If ξ is bijective on each closed neighborhood $N[v]$, then $\tilde{G}$ is called a *strongly deterministic graph* (or *SD-graph*) [8, 9]. In other words, in a D-graph there can be two adjacent vertices with the same label, but in a SD-graph there cannot be such vertices. The value of $\xi(V)$ for a vertex v is called a *label* of v. Note that the vertex labeling of D-graph is also called the injective labeling, first mentioned in [2]; the vertex labeling of SD-graph is also called distance-2 labeling, first mentioned in [1] (as a special case of p-distance labeling).

Let $\tilde{G_1} = (V_1, E_1, \Lambda, \xi_1, v')$ and $\tilde{G_2} = (V_2, E_2, \Lambda, \xi_2, v'')$ be D-graphs.

A bijective map $\varphi : V_1 \to V_2$ is called an isomorphism if $\varphi(v') = v''$, $\forall v \in V_1 \quad \xi_1(v) = \xi_2(\varphi(v))$, and $\forall v_1, v_2 \in V_1 \quad \{v_1, v_2\} \in E_1$ if and only if $\{\varphi(v_1), \varphi(v_2)\} \in E_2$. An isomorphism of D-graphs $\tilde{G_1}$ and $\tilde{G_2}$ is denoted by $\tilde{G_1} \cong \tilde{G_2}$.

In the following, the D-graphs considered in this article are studied using words in the alphabet of their labels. The set of all words of finite length in the alphabet Λ (which also includes the empty word ε of length 0) will be denoted by Λ^*. Let $p, q \in \Lambda^*$, then the concatenation of the words p and q will be denoted by pq. Let $p = v_1 \dots v_k$ be a walk in the D-graph $\tilde{G} = (V, E, \Lambda, \xi, v_0)$ and $\xi(v_1) = x_1, \dots, \xi(v_k) = x_k$. We will call the word $\sigma = x_1 \dots x_k$ a *walk-word* corresponding to $p = v_1 \dots v_k$ and denote it by $\sigma(p)$.

Let $\sigma = x_1 \dots x_k$. We call the word σ *valid* for the vertex $v_1 \in V$ if $\xi(v_1) = x_1$, and there exist vertices $v_2, \dots, v_k \in V$ such that $\xi(v_2) = x_2, \dots, \xi(v_k) = x_k$, and $\{v_1, v_2\}, \dots, \{v_{k-1}, v_k\} \in E$. In this case, we call the walk $p = v_1 v_2 \dots v_k$ a *word-walk* corresponding to the word σ and the vertex v_1, and denote it by $p(\sigma, v_1)$. We will denote the end vertex v_k of the walk $p(\sigma, v_1)$ by $v_1\sigma$. Note that since the function ξ is deterministic, and the validity of the word σ for the vertex v_1, it follows that the vertices $v_2, \dots, v_k$ are uniquely determined. We will denote the set of all words of finite length that are valid for the basepoint vertex v_0 of the D-graph $\tilde{G} = (V, E, \Lambda, \xi, v_0)$ by $\Lambda_0^*(G)$. Further, unless otherwise stated, we assume that all walks considered in this article and the words associated with them start from the basepoint vertex v_0. In the case of the validity of the word σ for the vertex v_0 (i.e., $\sigma \in \Lambda_0^*(G)$), we will denote the walk $p(\sigma, v_0)$ by $p_0(\sigma)$. The *inverse* of σ is denoted by σ^{-1} (i.e., $\sigma^{-1} = x_k \dots x_1$). We will call the word $\tau \in \Lambda^*$ a *prefix* of the word σ if there exists a word ρ such that $\tau\rho = \sigma$. We will denote this fact by $\tau \subseteq \sigma$; in this case, we will call ρ the *postfix* of σ. If $x'_i = x'_{i+2} (i+2 \leq k)$, then we will call the sequence $x'_i x'_{i+1} x'_{i+2}$ a *reverse*. If σ contains a reverse and $|\sigma| = 3$, then we will call σ a *reverse-word*.

Let $\sigma \in \Lambda^*$, $|\sigma| = k$ and $k > 1$. Let $m(\sigma)$ be equal to $\frac{k+1}{2}$ if k is odd and to $\frac{k}{2}$ if k is even. In other words, $m(\sigma)$ is the index in σ of the middle symbol of σ. Note that the notation $m(\sigma)$ will also be used in the third section of the article to define the branching vertices in transformation (5). Let $a(\sigma)$ denote the prefix of the word p of length $m(\sigma)$, and let $b(\sigma)$ denote the prefix of the word σ^{-1} of length $|\sigma| - m(\sigma)$. It is easy to see that $\sigma = a(\sigma)(b(\sigma))^{-1}$.

Let's fix a total linear order $<$ on Λ: $x_1 < x_2 < \ldots < x_p$. Let's define

the relation $\preceq$ on Λ^*:

1) for any $\sigma \in \Lambda^*$, we set $\sigma \preceq \sigma$;

2) for any $\sigma, \tau \in \Lambda^*$, if $|\sigma| < |\tau|$, then $\sigma \preceq \tau$;

3) let $|\sigma| = |\tau|$. For any $\sigma, \tau \in \Lambda^*$, if $\sigma = x'_1 \dots x'_s$, $\tau = x''_1 \dots x''_s$, $x'_1 = x''_1$, $\dots$, $x'_{k-1} = x''_{k-1}$, and $x'_k < x''_k$, then $\sigma \preceq \tau$.

We also define the relation $\precsim$ on Λ^*:

1) for any $\sigma \in \Lambda^*$, we set $\sigma \precsim \sigma$;

2) for any $\sigma, \tau \in \Lambda^*$, if $|\sigma| < |\tau|$, then $\sigma \precsim \tau$;

3) let $|\sigma| = |\tau|$. In the case of $\alpha(\sigma) \neq \alpha(\tau)$, if $\alpha(\sigma) = x'_1 \dots x'_m$, $\alpha(\tau) = x''_1 \dots x''_m$, $x'_1 = x''_1$, $\dots$, $x'_{k-1} = x''_{k-1}$, and $x'_k < x''_k$, then $\sigma \precsim \tau$. In the case of $\alpha(\sigma) = \alpha(\tau)$, if $\beta(\sigma) = x'_1 \dots x'_s$, $\beta(\tau) = x''_1 \dots x''_s$, $x'_1 = x''_1$, $\dots$, $x'_{k-1} = x''_{k-1}$, and $x'_k < x''_k$, then $\sigma \precsim \tau$.

In other words, according to the relations $\preceq$ and $\precsim$, words are first compared by their length. According to the relation $\preceq$, words of the same length are compared lexicographically with respect to the total order $<$. According the relation $\precsim$, for words of equal length, their prefixes are first compared lexicographically up to and including the middle. If these prefixes are equal then we compare lexicographically the words reversed to these postfixes up to the middle (not including the middle).

It is easy to see that the relations $\preceq$ and $\precsim$ are total, reflexive, antisymmetric, and transitive, i.e., each of these relations is a total order defined on Λ^* by the total order $<$ on the set Λ.

We will show that the order $\preceq$ is compatible with respect to concatenation.

Lemma 1. *Let $\sigma \preceq \tau$ and $\rho \in \Lambda^*$. Then $\rho\sigma \preceq \rho\tau$ and $\sigma\rho \preceq \tau\rho$.*

Proof. For the case $\rho = \varepsilon$, the proof is trivial: $\varepsilon\sigma = \sigma\varepsilon = \sigma \preceq \tau = \tau\varepsilon = \varepsilon\tau$. Let $\rho \neq \varepsilon$ and $\rho = x_1 \dots x_p$. Let $\sigma = \tau$. Then $\rho\sigma = \rho\tau$ and $\sigma\rho = \tau\rho$, so $\rho\sigma \preceq \rho\tau$ and $\sigma\rho \preceq \tau\rho$. Let $\sigma \neq \tau$. If $|\sigma| < |\tau|$, then $|\sigma\rho| = |\rho\sigma| < |\rho\tau| = |\tau\rho|$, so by point (2) of the definition of $\preceq$, we have $\rho\sigma \preceq \rho\tau$ and $\sigma\rho \preceq \tau\rho$. Now let $|\sigma| = |\tau|$, let $\sigma = x'_1 \dots x'_s$ and $\tau = x''_1 \dots x''_s$. From $\sigma \preceq \tau$, $\sigma \neq \tau$ and $|\sigma| = |\tau|$. From point (3) of the definition of $\preceq$, it follows that there exists such a k $(1 \leq k \leq s)$ that $x'_1 = x''_1$, $\dots$, $x'_{k-1} = x''_{k-1}$, and $x'_k < x''_k$. Then, since $x_1 = x_1, \dots, x_p = x_p$, $x'_1 = x''_1$, $\dots$, $x'_{k-1} = x''_{k-1}$, it follows from point (3) of the definition of $\preceq$ that $\rho\sigma = x_1 \dots x_p x'_1 \dots x'_s \preceq x_1 \dots x_p x''_1 \dots x''_s = \rho\tau$. Also, from point (3) of the definition of $\preceq$, it follows that $\sigma\rho = x'_1 \dots x'_s x_1 \dots x_p \preceq x''_1 \dots x''_s x_1 \dots x_p = \tau\rho$. $\square$

Note that the order $\precsim$, unlike the order $\preceq$, is not stable with respect to concatenation. Thus, for $\sigma_1 = 0134$, $\sigma_2 = 0115$, $\tau_1 = 0125$, $\tau_2 = 0234$, and $\rho = 77$, we have $\sigma_1 \precsim \tau_1$, $\sigma_2 \precsim \tau_2$, $\tau_1\rho \precsim \sigma_1\rho$, and $\rho\tau_2 \precsim \rho\sigma_2$.

2. Basic concepts of the linguistic presentation of deterministic graphs

This section outlines the basic concepts and algorithms related to the linguistic presentation of D-graphs. A significant part of this was published in [8] in fragmentary form; this article proposes an organized version of the linguistic presentation of D-graphs with some changes and additions. At the end of the section, we formulated the currently open questions related to the proposed linguistic presentation. Active work is currently underway to resolve these issues.

Let $\tilde{G} = (V, E, \Lambda, \xi, v_0)$ be a D-graph. We will call a *pair* $(C, L; x')$ two finite multisets of words $C \subset \Lambda^*$ and $L \subset \Lambda^*$ that satisfy the following conditions:

1. Any word from C begins and ends with the symbol x'.
2. Any word from L begins with the symbol x'.
3. The length of any word from C is greater than 2.
4. The length of any word from L is greater than 1.

We will call C and L as the *components* of the pair $(C, L; x')$. In cases where the value of the symbol x' is not essential in the context of the discussion, we will refer to the pair $(C, L; x')$ as (C, L).

Unlike [8], we define the components of a pair as multisets. This is because applying the pair transformations discussed in sections 3 and 4 of this article may result in duplicate elements in the components, in which case the result of the transformation is not formally a pair. At the same time, the functionality of the pair in terms of applying the algorithms discussed later in this article (the AP-algorithm and the compression algorithm) does not change: if (C_1, L_1) is the result of removing word repetitions from the components of the pair (C, L), and the result of the AP-algorithm with the pair (C, L) is undefined, then the result of the AP-algorithm with the pair (C_1, L_1) is also undefined, and if this result is defined, then it is the same for the pairs (C_1, L_1) and (C, L); the result of the compression algorithm is also the same.

In the linguistic presentation of the D-graph $\tilde{G}$, the pair $(C, L; x')$ defines the AP-algorithm, the result of which, for a given pair $(C, L; x')$, is either the D-graph $G(C, L; x')$ (this graph will also be denoted by $G(C, L)$), or a notification that the D-graph $G(C, L; x')$ does not exist. This algorithm uses the reduction algorithm as an auxiliary one, which transforms any graph G with labeled vertices into a deterministic graph $[G]$. The essence of reduction is the sequential identification (which we will refer to as *reduced merging*) of vertices with the same labels and edges incident to these vertices, in the case where such vertices have a common neighbor v; these reduced mergers are ordered by considering the vertices v in ascending order $\preceq$ from v_0 to v, taking into account the total linear order $<$, the reduction algorithm is given in [8].

The updated pseudocode for the AP-algorithm is presented below.

Note that step 4 is introduced to improve the speed of the algorithm, since the process of adding vertices and edges and applying the reduction algorithm to the result of such addition, caused by reprocessing the word $\sigma \in C \bigcup L$, will be exactly the same as in the first processing of the word σ. At the same time, removing duplicate elements in the components of a pair is obviously easier than adding additional vertices and edges to the graph and then reducing them by the reduction algorithm.

Example 1. Let $C = \{0120, 0311530, 031142530\}$, $L = \{0123, 03114230\}$. Figure 1(a), (b), and (c) show the graph obtained by applying step 5 to the first, second, and third words of the set C, respectively; Figure 1(d) and (e) show the graph obtained by applying step 6 to the first and second words of the set L, respectively. In these graphs, the initial vertex is highlighted in gray.

For the graph in Figure 1(e), the checks in steps 7, 8, and 9 are successful, so this graph is the result of the AP-algorithm.

Based on the AP-algorithm, we introduce two important definitions. We call the pair (C, L) *correct* if the result of executing the AP-algorithm on this pair is a D-graph $G(C, L)$. We will call a (correct) pair (C, L) is a *defining pair* for a D-graph $\tilde{G}$ if $\tilde{G} \cong G(C, L)$.

Let (C, L) be a defining pair for the D-graph $\tilde{G}$. Then the following conditions hold:

1. $\xi(v_0) = (x')$.
2. For every $\sigma \in C \bigcup L$ is performed $\sigma \in \Lambda_0^*(G(C, L))$.
3. For every $\sigma \in C$, the walk $p_0(\sigma)$ is closed.

Algorithm 1 AP-algorithm

Require: Pair $(C, L; x')$

Ensure: D-graph $G(C, L; x')$ or message that D-graph $G(C, L; x')$ does not exist

Let the graph $G(C, L; x')$ consists of a single vertex v_0 ▷ Step 1.
$\xi(v_0) \leftarrow x'$
if $C = \emptyset$ **and** $L = \emptyset$ **then** ▷ Step 2.
    return $G(C, L; x')$
end if
Order the words in C and L according to $\preceq$ ▷ Step 3.
Remove word repetitions in C and L ▷ Step 4.
for all $\sigma^i = x'x_1^i \dots x_n^i x' \in C$ **do** ▷ Step 5.
    Add vertices $v_1^i, \dots, v_n^i$ with labels $x_1^i, \dots, x_n^i$
    Add edges $\{v_0, v_1^i\}, \{v_1^i, v_2^i\}, \dots, \{v_{n-1}^i, v_n^i\}, \{v_n^i, v_0\}$
    Reduce the resulting graph
end for
for all $\sigma^j = x'x_1^j \dots x_n^j \in L$ **do** ▷ Step 6.
    Add vertices $v_1^j, \dots, v_n^j$ with labels $x_1^j, \dots, x_n^j$
    Add edges $\{v_0, v_1^j\}, \dots, \{v_{n-1}^j, v_n^j\}$
    Reduce the resulting graph
end for
for all $\sigma \in L$ **do** ▷ Step 7.
    if $v_0\sigma$ is not a leaf **or** $v_0\sigma = v_0$ **then**
        return "$G(C, L; x')$ does not exist"
    end if
end for
for all leaf vertex $v \neq v_0$ **do** ▷ Step 8.
    if there is no $\sigma \in L$ such that $v = v_0\sigma$ **then**
        return "$G(C, L; x')$ does not exist"
    end if
end for
for all $\sigma \in C \bigcup L$ **do** ▷ Step 9.
    if σ is not valid for v_0 **then**
        return "$G(C, L; x')$ does not exist"
    end if
end for
return $G(C, L; x')$

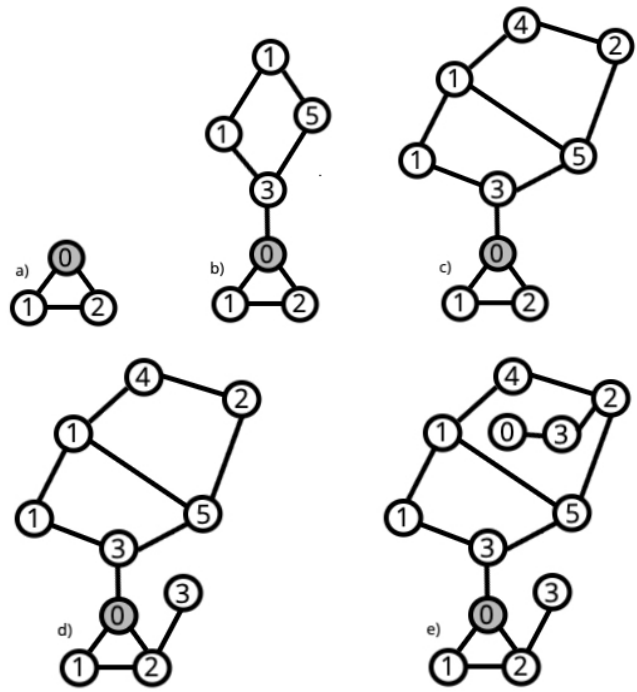


Figure 1. The execution process and result of the AP-algorithm

4. For every $\tau \in L$, the vertex $v_0\tau$ in the graph G is a leaf, and $v_0\tau \neq v_0$.

5. For every leaf vertex $v \in V$ that is different from the basepoint vertex v_0, there exists at least one word $\tau \in L$ such that $v_0\tau = v$.

6. For any pair $(C, L; x')$, either there is no presentation, or this presentation is uniquely determined.

Several different pairs may correspond to the same presentation. If the pair (C, L) is correct, then the graph $G(C, L)$ is the smallest inclusion graph that satisfies requirements (1) to (6) (i.e., no proper subgraph of the graph $G(C, L)$ satisfies these requirements simultaneously, in particular requirement (2)), so we will consider it a presentation for the pair (C, L).

We develop an algorithm for constructing a special pair, which was called canonical, by given D-graph $\tilde{G}$. To correctly represent this algorithm, we introduce the following concept. We call a *reachability basis*

of the D-graph $\tilde{G}$ the set of words $W = \{\sigma_0, \ldots, \sigma_{n-1}\}$ that satisfies the following three conditions:

1. For each vertex $v \in V$, there exists a word $\sigma \in W$ such that $v_0\sigma = v$.
2. For any $\sigma_i, \sigma_j \in W$ from $i \neq j$ follows $v_0\sigma_i \neq v_0\sigma_j$.
3. For any $\sigma \in W$, if $\sigma = \sigma' x (x \in \Lambda)$, then $\sigma' \in W$.

Note that the reachability basis of a D-graph is analogous to the reachability basis of an automaton, which is known from [10, 11].

For each vertex $v \in V$, let sp_v denote a word in Λ^* such that $v_0 sp_v = v$, and for any $\sigma \neq sp_v$ from $v_0\sigma = v$ follows $sp_v \preceq \sigma$. Note that from this definition it follows $sp_{v_0} = \xi(v_0)$. Let $\mathcal{V}_G = \{sp_{v_0}, \ldots, sp_{v_{n-1}}\}$. The following statement is true.

Theorem 1. *Let $\tilde{G} = (V, E, \Lambda, \xi, v_0)$ be a D-graph. Then $\mathcal{V}_G$ is a reachability basis of $\tilde{G}$.*

Proof. From the definitions of sp_v and $\mathcal{V}_G$, it follows that for each vertex $v \in V$ there exists a word $sp_v \in \mathcal{V}_G$ such that $v_0 sp_v = v$, and from $v_i \neq v_j$ it follows that $v_0 sp_{v_i} \neq v_0 sp_{v_i}$, i.e., the first and the second conditions of the reachability basis are satisfied. We will show that $\mathcal{V}_G$ satisfies the third condition of the reachability basis. Suppose by way of contradiction that this is not true, i.e., there exists such a $\sigma = \sigma' x (x \in \Lambda)$ in $\mathcal{V}_G$ that $\sigma' \notin \mathcal{V}_G$. Let $v_0\sigma' = v'$ and $v_0\sigma = v$. Let τ be a word from $\mathcal{V}_G$ such that $v_0\tau = v'$. Then, from the graph $\tilde{G}$ is a D-graph, it follows that $v_0\tau x = v$, and from the definition of $\mathcal{V}_G$, it follows that $\tau \preceq \sigma'$ and $\sigma = \sigma' x \preceq \tau x$, which contradicts Lemma 1. The resulting contradiction proves that $\mathcal{V}_G$ satisfies the third condition of the reachability basis. □

The spanning tree of the D-graph $\tilde{G}$, which is defined by the basis $\mathcal{V}_G$ (i.e., it is the union of paths $p_0(\sigma)$ for all $\sigma \in \mathcal{V}_G$), is denoted by $T(\mathcal{V}_G)$ or $T(G, v_0)$.

We continue with the pseudocode description of the AC-algorithm, which constructs a canonical defining pair (C_G, L_G) for a given D-graph $\tilde{G}$.

Example 2. Let's illustrate how the AC-algorithm works. Figure 2(a) shows the input D-graph $\tilde{G}$, with the basepoint vertex highlighted in grey; in Figure 2(b), the corresponding word from the set $\mathcal{V}_G$ is shown for each vertex of the graph.

After completing step 2, $\Lambda_G = \{023, 035230\}$. When applying step 3, the pairs of words $\sigma, \tau \in \mathcal{V}_G \setminus \Lambda_G$, neither of which is the prefix of the other and for which $v_0\sigma\tau^{-1} = v_0$, are 01 – 02, 035 – 0311, 0352 – 03114. Then, from this pairs, the words 0120, 0311530, 031142530 are added to Σ_G, that is, $\Sigma_G = \{0120, 0311530, 031142530\}$, $\Lambda_G = \{023, 035230\}$.

Algorithm 2 AC-algorithm

Require: D-graph $\tilde{G} = (V, E, \Lambda, \xi, v_0)$
Ensure: Pair (C_G, L_G)

$C_G \leftarrow \emptyset$
$L_G \leftarrow \emptyset$
if $|V| = 1$ **then** ▷ Step 1.
    return (C_G, L_G)
end if
for all $\lambda \in \mathcal{V}_G$ **do** ▷ Step 2.
    if $v_0\lambda$ is a leaf in $\tilde{G}$ **and** $v_0\lambda \neq v_0$ **then**
        $L_G \leftarrow L_G \bigcup \lambda$
    end if
end for
for all $\rho \in \mathcal{V}_G \setminus L_G$ **do** ▷ Step 3.
    for all $\tau \in \mathcal{V}_G \setminus L_G$ **do**
        if ρ is not a prefix of τ **and** τ is not a prefix of ρ **and** $v_0\rho\tau^{-1} = v_0$ **then**
            if $\rho\tau^{-1} \preceq \tau\rho^{-1}$ **then**
                $C_G \leftarrow C_G \bigcup \rho\tau^{-1}$
            else
                $C_G \leftarrow C_G \bigcup \tau\rho^{-1}$
            end if
        end if
    end for
end for
return (C_G, L_G)

We have proven that the pair (C_G, L_G) is defining for the graph $\tilde{G}$, the proof strategy is given in [8], this proof is quite large in size, so we do not give it in this work. Also, [8] presents the metric properties of the power and volume of the canonical defining pair (C_G, L_G). Further in this article, we will show that the pair (C_G, L_G) is the smallest pair in all parameters that is defining for the D-graph $\tilde{G}$. The article will also

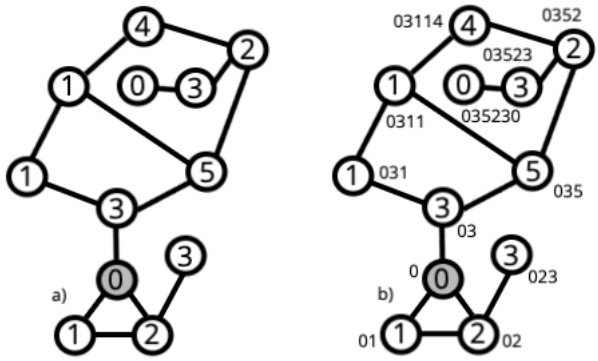


Figure 2. An example of the AC-algorithm

demonstrate the decisive role of the canonical defining pair in solving the problem of characterizing a D-graph.

Let's introduce definitions of concepts related to the canonical defining pair, which will be used further in this article. We will call the vertices $v_0\sigma$ and $v_0\tau^{-1}$ the *generators* for the word $\sigma\tau^{-1}$ or $\tau\sigma^{-1}$ that was added to C_G in step 3 of the AC-algorithm. Let v', v'' be the generators of $\sigma \in C_G$. Then, since each of the words $sp_{v'}$ and $sp_{v''}$ is not a prefix of the other, and $\{v', v''\} \in E$, it follows that $|\,|sp_{v'}|-|sp_{v''}|\,| \leq 1$. We call the word $\sigma \in \Lambda^*$ a *canonical word* for $\tilde{G}$ if $a(\sigma) \in \mathcal{V}_G$, $b(\sigma) \in \mathcal{V}_G$, $v_0\sigma = v_0$, and σ does not contain any reverses.

Lemma 2. *Let σ be a canonical word for the graph $\tilde{G}$. If $|\sigma|$ is an even number, then σ^{-1} is also a canonical word for the graph $\tilde{G}$; if $|\sigma|$ is an odd number, then σ^{-1} is not a canonical word for the graph $\tilde{G}$.*

Proof. Let σ be a canonical word for graph $\tilde{G}$ and $|\sigma|$ be an even number. Then, by the definitions of $a(\sigma)$ and $b(\sigma)$, the equalities $a(\sigma^{-1}) = b(\sigma)$ and $b(\sigma^{-1}) = a(\sigma)$ hold, so $\sigma^{-1} = b(\sigma)(a(\sigma))^{-1}$, so σ^{-1} is a canonical word for the graph $\tilde{G}$. Now let $|\sigma|$ be an odd number. Then, by the definitions of $a(\sigma)$ and $b(\sigma)$, $v_0a(\sigma) = v_0a(\sigma^{-1})$. Since σ is a canonical word for $\tilde{G}$, it follows that $a(\sigma) \in \mathcal{V}_G$, therefore $a(\sigma^{-1}) \notin \mathcal{V}_G$, i.e., σ^{-1} is not a canonical word for the graph $\tilde{G}$. □

Lemma 3. *C_G consists of all canonical words for the graph $\tilde{G}$ whose length is an odd number and all canonical words σ for $\tilde{G}$ whose length is an even number such that $\sigma \preceq \sigma^{-1}$.*

Proof. Necessity. Let $\sigma \in C_G$, v', v'' are the generators of σ. We will prove that σ is a canonical word for $\tilde{G}$. Let $sp_{v'} = \rho$, $sp_{v''} = \tau$, and $\sigma = \rho\tau^{-1}$. According to the AC-algorithm, σ does not contain reverses. Let $|\sigma|$ be an odd number. Considering the inequality $| \, |sp_{v_1}| - |sp_{v_2}| \, | \leq 1$, one of two equalities holds: $|\rho| = |\tau| + 1$ or $|\tau| = |\rho| + 1$. Suppose that the equality $|\tau| = |\rho| + 1$ holds. Let $\tau = \tau' x (x \in \Lambda)$. Then $|\rho| = |\tau'|$. Since $sp_{v''} = \tau' x$ and $\{v_0\rho, v''\} \in E$, then $\tau' x \preceq \rho x$, so $\tau' \preceq \rho$, in which case $\tau\rho^{-1} = \tau' x \rho^{-1} \preceq \rho x \tau^{-1} = \rho\tau^{-1}$, which contradicts $\rho\tau^{-1} \in C_G$. Therefore, the equality $|\rho| = |\tau| + 1$ holds. In this case, $a(\sigma) = \rho$, $b(\sigma) = \tau^{-1}$, therefore σ is a canonical word for $\tilde{G}$. Now let $|\sigma|$ be an even number. Then $|\rho| = |\tau|$, so $a(\sigma) = \rho$, $b(\sigma) = \tau^{-1}$, i.e., σ is a canonical word for $\tilde{G}$, and $\sigma \preceq \sigma^{-1}$.

The sufficiency of the statement follows directly from step 3 of the AC-algorithm. Let σ be a canonical word for $\tilde{G}$. Then, from $a(\sigma), b(\sigma) \in \mathcal{V}_G$, it follows that neither of the words $a(\sigma), b(\sigma)$ is a prefix of the other. Since $v_0\sigma = v_0$, if $|\sigma|$ is an odd number, then $\sigma = a(\sigma)(b(\sigma))^{-1} \in C_G$. If $|\sigma|$ is an even number, then one of the two words $a(\sigma)(b(\sigma))^{-1}$ or $b(\sigma)(a(\sigma))^{-1}$, which is smaller in order $\preceq$, is added to C_G. □

Thus, the developed linguistic presentation of D-graphs, which uses only two (multi)sets of string-type data, is effective for the D-graph representation, as it simplifies the structure compared to matrix or dictionary implementations of this representation. This approach improves integration with text data types and forms a more intuitive, user-oriented graph description interface. At the same time, a number of problems and tasks related to the proposed linguistic presentation of D-graphs remain unresolved. In our opinion, the most important problems and tasks are as follows:

- Is there any connection between graph G and its reduction $[G]$, other than the fact that $[G]$ is the result of applying the reduction algorithm to graph G (i.e., is it possible to characterize $[G]$ in terms of G without performing the reduction algorithm)?

- Will the result of the reduction algorithm change if the reduced vertex mergers are performed in arbitrary order, as opposed to the order specified in [8]?

- Will the result of the AP-algorithm change if steps 5 and 6 are performed for words from C and L arranged in arbitrary order, rather than ordered by $\preceq$ as specified in this article?

- What is the complexity of the AP-algorithm and the reduction algorithm?
- Is there a relationship between the structure of D-graphs and the associated relational structures defined by their endomorphisms and automorphisms? Such relationships for hypergraphs were established in [12].

At the same time, we see a number of practical applications of the proposed linguistic presentation of D-graphs, one of which (a characterization of linguistic presentation) will be solved in the following sections.

3. Equivalent transformations of a correct pair

Next, we consider the solution to the problem of characterizing a given correct pair (C, L) for a given deterministic graph D-graph $\tilde{G}$, which we understand as determining whether the pair (C, L) is defining for $\tilde{G}$ without constructing the graph $G(C, L)$ using the AP-algorithm, and then checking for the presence or absence of isomorphism between the graphs $\tilde{G}$ and $G(C, L)$. To solve the characterization problem, equivalent transformations of a correct pair are introduced, the essence of which is understood in terms of the isomorphism of the graph constructed from the initial pair with the graph constructed from the pair resulting from the transformations. Based on these transformations, we develop the compression algorithm, which will be discussed in the next section of the article. In this section, we have presented only those equivalent transformations of a correct pair that are used by the compression algorithm and whose equivalence has been proven.

We will call the correct pairs $(C, L; x')$ and $(C_1, L_1; x')$ *equivalent* if $G(C, L; x') \cong G(C_1, L_1; x')$. We will denote the equivalence of the pairs $(C, L; x')$ and $(C_1, L_1; x')$ by $(C, L; x') \sim (C_1, L_1; x')$. Let's consider some elementary transformations of a correct pair that lead to a pair equivalent to it, where the resulting pair is in some sense simplified relative to the source pair (in the resulting pair, either the number of words is reduced, or the length of the words is reduced, or some words are replaced by words that are smaller in order $\preceq$ or $\precsim$). These transformations will form the basis of the compression algorithm steps, which will be presented in the next section of this work. Next, we denote the initial correct pair by $(C, L; x')$.

1) Removing of word repetitions in C and L. Let $\sigma \in C \bigcup L$, and σ be repeated several times in one of the components of the pair (C, L)

(from the correctness of (C, L) it follows that $\sigma \notin C \bigcap L$). Let the pair (C_1, L_1) be the result of removing repetitions of the word σ from the corresponding component of the pair (C, L); further, we will denote this result as $DEL-REP(\sigma, (C, L))$, performing this transformation reduces the number of words in the corresponding component.

2) Removing reverse-words from C. Let $\sigma \in C$, $\sigma = xyx$ $(x, y \in \Lambda)$, and C_1 be the result of removing σ from C. Let's denote the result of this transformation as $DEL-RW(\sigma, (C, L))$; performing this transformation reduces the number of words in the first component.

3) Removing reverses from words of C and L. Let $\sigma \in C \bigcup L$, $|\sigma| > 3$, and σ contain one or more reverses. Let xyx be the first reverse on the left of the word σ, and let $\sigma = \rho_1 xyx \rho_2$, where $\rho_1, \rho_2 \in \Lambda^*$. Let $\sigma' = \rho_1 x \rho_2$. Let the pair (C_1, L_1) be formed from the pair (C, L) by replacing in the corresponding component word σ with the word σ'. Let the result of this transformation be denoted by $REM - R(\sigma, (C, L))$; performing this transformation reduces the length of the words in the corresponding component. Note that this transformation in some sense is similar to the transformation of a free semigroup which is used during the description of verbal congruences on free trioids [13, 14].

4) Inverting words in C. Let $\sigma \in C$, $\sigma^{-1} \preceq \sigma$, and C_1 be the result of replacing in C the word σ with the word σ^{-1}. Let's denote the result of this transformation as $ORD(\sigma, (C, L))$; as a result of this transformation, the resulting pair differs from the source by one word, and this word in the resulting pair is less than the transformed word in the order $\preceq$.

5) Optimization of paths according to the branching vertex.

Let σ contain no reverses and $\sigma \in C$. Let also $\sigma' \neq \sigma$, $\sigma' \neq \varepsilon$, $\sigma' \subseteq \sigma$, $\sigma' = \sigma_1 x$ $(x \in \Lambda)$. We will refer to the vertex $v = v_0 \sigma'$ as the *branch vertex*. Let $\sigma = \sigma_1 x \sigma_2$. Then, with respect to the path from the basepoint vertex to the branching vertex, one of two mutually exclusive cases is possible when comparing the words σ_1 and $(\sigma_2)^{-1}$ in $\preceq$:

1) $\sigma_1 \preceq (\sigma_2)^{-1}$.

a) Let $\tau \neq \sigma$, $\tau \in C \bigcup L$, and $(\sigma_2)^{-1} x$ be the prefix of τ, i.e., $\tau = (\sigma_2)^{-1} x \tau'$ for some $\tau' \in \Lambda^*$. Let $\tau_1 = \sigma_1 x \tau'$. Let (C_1, L_1) denote the pair in which the word τ is replaced by the word τ_1 in the corresponding component.

b) Let $\tau \neq \sigma$, $\tau \in C$, and τ end with $x\sigma_2$, i.e., $\tau = \tau' x \sigma_2$ for some $\tau' \in \Lambda^*$. Let $\tau_2 = \tau' x (\sigma_1)^{-1}$. Let C_2 be the result of replacing the word τ with the word τ_2 in C.

2) $(\sigma_2)^{-1} \preceq \sigma_1$.

a) Let $\tau \neq \sigma$, $\tau \in C \bigcup L$, and $\sigma_1 x$ be the prefix of τ, i.e., $\tau = \sigma_1 x \tau'$ for some $\tau' \in \Lambda^*$. Let $\tau_3 = (\sigma_2)^{-1} x \tau'$. Let (C_3, L_3) denote the pair in which the word τ is replaced by the word τ_3 in the corresponding component.

b) Let $\tau \neq \sigma$, $\tau \in C$, and τ end with $x(\sigma_1)^{-1}$, i.e., $\tau = \tau' x(\sigma_1)^{-1}$ for some $\tau' \in \Lambda^*$. Let $\tau_4 = \tau' x \sigma_2$. Let C_4 be the result of replacing the word τ with the word τ_4 in C.

As a result of this transformation, the resulting pair differs from the source pair by one word. In the first and the third cases, this word in the resulting pair is less than the transformed word in the order $\preceq$, and in the second and the fourth cases, this word in the resulting pair is less than the transformed word in the order $\precsim$.

We will show that each of the specified transformations is an equivalent transformation.

Theorem 2. *Let (C, L) be a correct pair and let (C', L') be obtained from (C, L) by a single application of any of the elementary transformations (1) – (5). Then $(C, L) \sim (C', L')$.*

Proof. Let $\sigma \in C \bigcup L$ and $(C', L') = DEL - REP(\sigma, (C, L))$. Then, from step 4 of the AP-algorithm, it follows that $(C, L) \sim (C', L')$.

Let $\sigma \in C$, $\sigma = xyx$ $(x, y \in \Lambda)$ and $(C', L') = DEL - RW(\sigma, (C, L))$. From the definition $DEL - RW$, it follows that $L' = L$. When processing the AP-algorithm of the word σ, a vertex with label y and an edge $\{v_0, v_0 xy\}$ are added to the already constructed graph fragment, and, in the case where the vertex $v_0 xy$ already existed in the previously constructed graph fragment, reductions of the resulting graph. We will show that in one of the components of the pair (C, L) there must exist a word $\rho \neq xyx$, for which the fifth or sixth step of the AP-algorithm constructs the vertex $v_0 xy$ and the edge $\{v_0, v_0 xy\}$. Conversely, suppose that such a word does not exist. In this case, no vertex can be merged with vertex $v_0 xy$, so in graph $G(C, L)$, vertex $v_0 xy$ must be leaf. At the same time, from the correctness of the pair (C, L), and, accordingly, the feasibility of the eighth step of the AP-algorithm when constructing the graph $G(C, L)$, it follows that there exists in L such a word τ that $v_0 \tau = v_0 xy$, where the last symbol of the word τ is y, and the penultimate symbol is x. Then, given that the leaf vertex $v_0 xy$ is adjacent only to vertex v_0, when processing the word ρ with the AP-algorithm, a vertex with label y and an edge between this vertex and basepoint vertex v_0 must be constructed; when performing the reduction, a reduced merging of this vertex with vertex $v_0 xy$ and the edge between this vertex and v_0 will be performed with edge $\{v_0, v_0 xy\}$; this contradicts the assumption

that word ρ does not exist. Therefore, in one of the components of the pair (C, L), and therefore in one of the components of the pair (C', L), there is a word according to which the edge $\{v_0, v_0xy\}$ is constructed. In this case, the result of the AP-algorithm with the pair (C, L) is identical to the result of the AP-algorithm with the pair (C', L), so the graph $G(C', L)$ exists, and $G(C, L) \cong G(C', L)$.

Let $\sigma \in C \bigcup L$, $(C', L') = REM - R(\sigma, (C, L))$, $\sigma = \rho_1 xyx\rho_2$, $\sigma' = \rho_1 x\rho_2$, and (C', L') is the result of replacing the word σ with the word σ' in the corresponding component (C, L). Let $\sigma \in C$, then $L' = L$. The process of executing the AP-algorithm with the pair (C, L) differs from that with the pair (C', L) in the processing of the words σ and σ' in the fifth step of the AP-algorithm. The result of this processing after further reduction for the word σ differs from the similar result for the word σ' by the presence in the first case of the vertex $v_0\rho_1 xy$ and the edge $\{v_0\rho_1 x, v_0\rho_1 xy\}$, all other additions of vertices and edges, as well as reduced mergers in both cases, are the same. As with the $DEL - RW$ transformation, the presence of the word $\rho \neq \sigma$ in one of the components of the pair (C, L) must be proven, after which the fifth or sixth step of the AP-algorithm constructs the edge $\{v_0\rho_1 x, v_0\rho_1 xy\}$, from which follows the existence of the graph $G(C', L)$ and $G(C, L) \cong G(C', L)$, i.e., $(C, L) \sim (C_1, L)$. In the case of $\sigma \in L$, the equivalence $(C, L) \sim (C, L')$ holds, the proof is similar to the case of $\sigma \in C$.

Let $\sigma \in C$ and $(C', L') = ORD(\sigma, (C, L))$. From the definition of the ORD transformation, it follows that $L' = L$. The process of executing the AP-algorithm with the pair (C, L) differs from that with the pair (C', L) in the processing of the words σ and σ^{-1} in the fifth step of the AP-algorithm. In this processing, all additions of vertices and edges, as well as reduced mergers, are the same in both cases, so $(C, L) \sim (C', L)$.

Let (C', L') be the result of applying transformation (5) to the pair (C, L). Let case (1a) of this transformation be true, i.e., $\exists \sigma = \sigma_1 x\sigma_2 \in C$, $\sigma_1 \preceq (\sigma_2)^{-1}$, $\tau = (\sigma_2)^{-1} x\tau'$, $\tau_1 = \sigma_1 x\tau'$, and (C', L') is the result of replacing the word τ with the word τ_1 in the corresponding component (C', L'). The process of executing the AP-algorithm with the pair (C, L) differs from that with the pair (C', L') in the processing of the words τ and τ_1. When processing the word τ in the fifth or the sixth step of the AP-algorithm, vertices and edges corresponding to the symbols of the word $(\sigma_2)^{-1}$, vertices and edges corresponding to the symbols of the word τ', as well as the branching vertex, the edge between this vertex and the vertex $v_0(\sigma_2)^{-1}$, and the edge between the branching vertex and the

vertex $v_0\tau'$. When processing the word τ_1 in the fifth or the sixth step of the AP-algorithm, vertices and edges corresponding to the symbols of the word σ_1, vertices and edges corresponding to the symbols of the word τ', as well as the branching vertex, the edge between this vertex and the vertex $v_0\sigma_1$, and the edge between the branching vertex and the vertex $v_0\tau'$. Next, as a result of the reduction of the resulting graph, in the first case, there is a reduced merging of vertices and edges corresponding to the symbols of the word $(\sigma_2)^{-1}$, as well as the branching vertex with the corresponding vertices that were added when processing the word σ. In the second case, as a result of reduction, there is a reduced merging of vertices corresponding to the symbols of the word σ_1, as well as the branching vertex with the corresponding vertices that were added when processing the word σ. All other additions of vertices and edges, as well as further reduction when processing the words τ and τ_1, are identical, so the results of processing the words τ and τ_1 by the AP-algorithm coincide, i.e., $(C, L) \sim (C', L')$. The equivalence $(C, L) \sim (C', L')$ in the other three cases of transformation (5) is proven similarly. □

All of these elementary transformations are key to the compression algorithm described in the next section of this article.

4. The compression algorithm

This section presents a compression algorithm developed on the basis of elementary transformations of a correct pair considered in the previous section. Using the compression algorithm, we propose a solution to the characterization problem for a given correct pair (C, L) and D-graph $\tilde{G}$, which consists of performing compression (C, L) and comparing its result with the canonical defining pair (C_G, L_G). In our opinion, the proposed solution to the characterization problem is appropriate given two key advantages: first, it remains human-oriented, allowing the use of ordinary text strings as an interface for data interaction with the algorithm for their transformation; second, it is advantageous from a technical point of view, as it represents data in the form of a regular string type, which greatly simplifies its processing and integration into computing systems. At the same time, the compression algorithm is in a sense similar to the algorithm that is a solution to the problem of characterizing the system of defining relations for a finite automaton without outputs [15].

Given that the transformations (1) – (4) are simpler in the algorithmic terms than the transformation (5), the main focus in developing the

compression algorithm was on optimizing the transformation (5). We will show that, taking into account the possibility of performing the transformations (1) – (4) after each execution of the transformation (5), in the compression algorithm for the transformation (5), as branch vertices for the word $\sigma = x'_1x'_2\dots x'_{k-1}x'_k \in C$ ($x'_i \in \Lambda$, $x'_1 = x'_k = x'$) it is sufficient to take the vertices $v_0x'_1\dots x'_{m(\sigma)}$ and $v_0x'_1\dots x'_{m(\sigma)+1}$ (hereinafter referred to as vertices with indices $m(\sigma)$ and $m(\sigma)+1$, respectively).

Suppose that case (1a) is true for the elementary transformation 5, i.e. $\sigma \in C$, $\sigma = \sigma_1x\sigma_2$, where $v_0\sigma_1x$ is the branching vertex, $\sigma_1 \preceq (\sigma_2)^{-1}$, $\tau \neq \sigma$, and $\tau = (\sigma_2)^{-1}x\tau'$ for some $\tau' \in \Lambda^*$. Let i denote the index of the character corresponding to the branching vertex in the word σ. From $\sigma_1 \preceq (\sigma_2)^{-1}$ and the previous elementary transformation ORD, it follows that $1 \leq i \leq m(\sigma)$. As a result of executing the transformation (5), we obtain the replacement of the word τ with the word $\rho = \sigma_1x\tau'$.

We will show that in the case $i \neq m(\sigma)$, as a result of performing the elementary transformation 5 at the branching vertex with index $m(\sigma)$ and a certain sequence of other elementary transformations, we obtain the same result of transformation (replacement) of τ. Let's introduce the following notation. Let θ denote the fragment of σ from the symbol following the branching vertex in the previous case to the symbol with index $m(\sigma)$ (we denote this symbol by $\sigma(m(\sigma))$), not including the symbol with index $m(\sigma)$ itself, and let ϑ denote the fragment of σ from the symbol with index $m(\sigma)+1$ to the end of the word σ, i.e., $\sigma = \sigma_1x\theta\sigma(m(\sigma))\vartheta$. In these notations, $\tau = \vartheta^{-1}\sigma(m(\sigma))\theta^{-1}x\tau'$. The sequence ϑ^{-1} in the word τ is replaced by $\sigma_1x\theta\sigma(m(\sigma))$, i.e., the word τ is replaced by the word $\tau_1 = \sigma_1x\theta\sigma(m(\sigma))\theta^{-1}x\tau'$. Next, after removing the reverses from the word τ_1 in a quantity equal to $|\theta|$, we obtain the transformation of the word τ_1 into the word $\tau_2 = \sigma_1xzx\tau'$, where z is the first character of the word θ; in the case where θ is an empty word, then $z = \sigma(m(\sigma))$, after which the reverse xzx is removed from τ_2, i.e., we obtain the transformation of the word τ_2 into the word $\tau_3 = \sigma_1x\tau'$, i.e., $\tau_3 = \rho$.

Thus, it is shown that under the condition of performing the elementary transformations $REM-R$ and ORD (the transformation $REM-R$ is performed both before and after the transformation (5)), for the transformation (5) in case (1a), the branch vertex in the word σ can be taken as the vertex with index $m(\sigma)$. Let $OPT - LB(\sigma,\tau)$ denote the result of transforming the word τ at the branching vertex of the word σ corresponding to the index $m(\sigma)$; in our case, $OPT - LB(\sigma,\tau) =$

$\sigma_1 x\theta\sigma(l(\sigma))\theta^{-1}x\tau'$. In the computational terms, the disadvantage of such a transformation is the additional, possibly repeated, use of the transformation $REM - R$. The advantage is that the branching vertex is determined unambiguously, i.e., to determine it, it is not necessary to use a loop from the first character of the word σ to the character with index $m(\sigma)$ and then consider all words from the components C and L.

For the case (1b), the result of the transformation according to the branching vertex with index i in the word σ $(1 \le i < m(\sigma))$ will be equal to the same transformation according to the branching vertex with index $m(\sigma)$ followed by the possible repeated execution of the transformation $REM - R$ (this is proved in a dual way to the case 1(a)). Let's denote this transformation corresponding to the branching vertex with index $m(\sigma)$ by $OPT - LE(\sigma, \tau)$.

Let case (2a) hold for the elementary transformation (5), i.e., $\sigma \in C$, $\sigma = \sigma_1 x\sigma_2$, where $v_0\sigma_1 x$ is the branching vertex, $(\sigma_2)^{-1} \preceq \sigma_1$, $\tau \ne \sigma$, and $\tau = \sigma_1 x\tau'$ for some $\tau' \in \Lambda^*$. Let i denote the index of the symbol corresponding to the branching vertex in the word σ. From $(\sigma_2)^{-1} \preceq \sigma_1$ and the previous execution of the elementary transformation ORD, it follows that $m(\sigma) + 1 \le i \le |\sigma|$. Similar to case (1a), the result of performing the transformation (5) with the word τ in the case $i \ne m(\sigma)+1$ will be the same as the result of performing it in the case $i = m(\sigma) + 1$ with the subsequent, possibly repeated application of the transformation $REM - R$. Let's denote the transformation corresponding to the branching vertex with index $m(\sigma) + 1$ as $OPT - RB(\sigma, \tau)$.

Similarly, for case (2b), the result of performing the transformation (5) with the word τ in the case $i \ne m(\sigma) + 1$ will be the same as the result of performing it in the case $i = m(\sigma)+1$ with subsequent, possibly repeated, application of the $REM - R$ transformation to it; Let's denote the transformation corresponding to the branching vertex with index $m(\sigma) + 1$ as $OPT - RE(\sigma, \tau)$.

In these transformations the symbols L / R (left / right) denote one of two possible indices of the branching vertex, and the symbols B / E (begin / end) denote the direction of replacement. From the transformations $OPT - LB(\sigma, \tau)$, $OPT - LE(\sigma, \tau)$, $OPT - RB(\sigma, \tau)$, $OPT - RE(\sigma, \tau)$, we created a composite operation $PATH - OPT(C, L)$, it is shown in Figure 3.

Let's define a compression algorithm that transforms a correct pair $(C, L; x')$ into $< C, L; x' >$ (or simply $< C, L >$ when x' is irrelevant). Its components are denoted by $< C >$ and $< L >$. This algorithm is an iterative process $(C_0, L_0) \to (C_1, L_1) \to \ldots \to (C_n, L_n)$, in which

```
DEFINE PATH-OPT(C, L):
    FOR σ IN C:
        FOR τ IN C:
            OPT-LB(σ, τ)
            OPT-RB(σ, τ)
            OPT-LE(σ, τ)
            OPT-RE(σ, τ)
        FOR τ IN L:
            OPT-LB(σ, τ)
            OPT-RB(σ, τ)
```

Figure 3. The notation of the operation $PATH - OPT(C, L)$

$(C_0, L_0) = (C, L)$, $(C_n, L_n) =< C, L >$, the pair resulting from the current iteration is the input argument for the next iteration. Given this, in the pseudocode description of the compression algorithm, we denote the name of the pair that is an argument in operations by (C', L').

Algorithm 3 Compression algorithm

Require: Pair (C, L)
Ensure: Pair $< C, L >$

repeat
  $SORT(C', L')$
  $DEL - REP(\tau, C', L'))$
  $DEL - RW(\tau, C', L'))$
  if $REM - R(\tau, (C', L'))$ **then**
    continue
  end if
  if $ORD(\tau, (C', L'))$ **then**
    continue
  end if
  if $PATH - OPT(C', L')$ **then**
    continue
  end if
  exit
until false
return $< C, L >$

Since each operation of the compression algorithm, except sorting, either reduces the number of words in a pair component, or reduces the length of some word, or replaces it with a smaller word in $\preceq$ or $\precsim$, for a finite source pair, the compression algorithm completes its work with a uniquely determined result in a finite number of steps. Figure 4 shows

an example of the compression algorithm.

C = {030, 031142530, 035142530, 0120311530}, L = {0123, 03115230, 03514230}

01. **Operation DEL-RW(τ) is performed; τ = 030**
02. **Operation OPT-LE(σ, τ) is performed; σ = 031142530, τ = 035142530, modified = 035141130**
03. **Operation REM-R(τ) is performed; τ = 035141130, modified = 0351130**
04. **Operation ORD(τ) is performed; τ = 0351130, modified = 0311530**
05. **Operation OPT-LE(σ, τ) is performed; σ = 0311530, τ = 0120311530, modified = 0120311130**
06. **Operation REM-R(τ) is performed; τ = 0120311130, modified = 01203130**
07. **Operation REM-R(τ) is performed; τ = 01203130, modified = 012030**
08. **Operation REM-R(τ) is performed; τ = 012030, modified = 0120**
09. **Operation OPT-RB(σ, τ) is performed; σ = 0120, τ = 0123, modified = 023**
10. **Operation OPT-RB(σ, τ) is performed; σ = 0311530, τ = 03115230, modified = 035230**
11. **Operation OPT-LB(σ, τ) is performed; σ = 0311530, τ = 03514230, modified = 03114230**
12. **Operation OPT-LB(σ, τ) is performed; σ = 031142530, τ = 03114230, modified = 035230**
13. **Operation DEL-REP(τ) is performed; τ = 035230**

< C > = {0120, 0311530, 031142530}; < L > = {023, 035230}

Figure 4. An example of the compression algorithm

The criterion for solving the problem of characterizing a correct pair, which was defined at the beginning of previous section, is the following statement.

Theorem 3. *Let $\tilde{G}$ is some D-graph, and (C, L) be the correct pair. Then (C, L) is a defining pair for $\tilde{G}$ if and only if $< C, L >= (C_G, L_G)$.*

Proof. Sufficiency. As mentioned above, the execution of the compression algorithm can be represented as an iterative process $(C_0, L_0) \to (C_1, L_1) \to \ldots \to (C_n, L_n)$, in which $(C_0, L_0) = (C, L)$, $(C_n, L_n) = < C, L >$, the pair resulting from the current iteration is the input argument for the next iteration. According to Theorem 2, each transformation transforms a pair into a pair equivalent to it (transformations $OPT-LB(\sigma, \tau)$, $OPT-LE(\sigma, \tau)$, $OPT-RB(\sigma, \tau)$, $OPT-RE(\sigma, \tau)$, which are parts of the composite operation $PATH-OPT(C, L)$, are special cases of the elementary transformation (5)), and the operation $SORT(C, L)$ does not change the graph $G(C, L)$. Therefore, $\tilde{G} = G(C_n, L_n) = \ldots = G(C_0, L_0) = G(C, L)$, i.e. (C, L) is a defining pair for $\tilde{G}$.

Necessity. Let $(C, L; x')$ be a defining pair for $\tilde{G}$ and v_0 be the base-point vertex of the graph $\tilde{G}$. Let's prove the auxiliary statement

Lemma 4. *Let $< C >$, $< L >$ be the components of the compression result of the correct pair $< C, L; x' >$. Then for any words $\sigma \in < C >$, $\rho \in < L >$ with $\tau_1 \subseteq a(\sigma)$, it follows that $sp_{v_0\tau_1} = \tau_1$, from $\tau_2 \subseteq b(\sigma)$ follows $sp_{v_0\tau_2} = \tau_2$, and from $\tau_3 \subseteq \rho$ follows $sp_{v_0\tau_3} = \tau_3$ in the graph $\tilde{G}$.*

Proof. We will prove this by induction on the length i of the prefix of the corresponding word. Let $i = 1$. Since for any prefix τ of length 1 of words $a(\sigma)$, $b(\sigma)$, ρ, we have $\tau = x'$, $v_0x' = v_0$, and $sp_{v_0} = x'$, then $sp_{v_0\tau} = \tau$, i.e., the induction base is satisfied.

Suppose that the Lemma holds for $i \le k$. We will show that it also holds for $i = k + 1$. Contradiction: suppose that this is not the case, i.e., there exists τ that is the prefix of either $a(\sigma)$, or $b(\sigma)$, or ρ for some $\sigma \in < C >$, $\rho \; in < L >$, such that $|\tau| = k + 1$ and τ is not the shortest word according to $\preceq$ defining the path from vertex v_0 to vertex $v_0\tau$. Then one of three cases holds: a) there exists such a $\sigma \in < C >$ that $\tau \subseteq a(\sigma)$; b) there exists such a $\sigma \in < C >$ that $\tau \subseteq b(\sigma)$; c) there exists such a $\rho \in < L >$ that $\tau \subseteq \rho$. Let case (a) hold.

Let $\sigma = \tau\sigma'$, then from $\tau \subseteq a(\sigma)$ it follows that $|\sigma| \ge 2k + 1$. Let $v_0\tau = v$, $\xi(v) = x$, $\tau = \tau'x$, $v_0\tau' = v'$, $\xi(v') = x'$. Then $|\tau'| = k$, by the induction assumption $\tau' = sp_{v'}$. Let $\theta = \theta'x = sp_v$, then, taking into account the induction assumption, $|\theta'x| \le k + 1$, therefore $|\theta'| \le k$; let $v_0\theta' = v''$. From $\theta = sp_v$ it follows that $\theta' = sp_{v''}$ and $\theta' \preceq \tau'$. From $\tau'x \ne \theta'x$ it follows that $\tau' \ne \theta'$, therefore, taking into account $\tau' = sp_{v'}$, $\theta' = sp_{v''}$, the assumption of induction, and the determinism of the graph $G(< C, L >)$ hold, $\xi(v') \ne \xi(v'')$, $v' \ne v''$, and the walk $p_0(\tau'x(\theta')^{-1})$ is closed. At the same time, from $\tau' = sp_{v'}$, $|\tau'| = k$, $v_0\theta x' = v'$ it follows that $|\theta x'| \ge k$, therefore $k-2 \le |\theta'| \le k$. Since $< C, L >$ is a defining pair of the graph $G(< C, L >)$, there exists ϑ such that $v_0\vartheta = v''$, and for some $\vartheta' \in \Lambda^*$ at least one of the statements $\vartheta x\vartheta' \in < C >$ or $\vartheta' x\vartheta^{-1} \in < C >$ holds. From the induction assumption, taking into account $|\theta'| \le k$, it follows that $\vartheta = \theta'$, i.e., either $\theta' x\vartheta' \in < C >$ or $\vartheta' x(\theta')^{-1} \in < C >$.

Then, from the fact that all possible compression operations were performed on the pair $< C, L >$, in graph $G(< C, L >)$, the vertex $v_0(\tau)$ can be identified with the vertex $v_0(\theta)$ only if one of the two words $\theta' x(\tau')^{-1}$ or $\tau' x(\theta')^{-1}$ exists in $< C >$, which is smaller than the other in the order $\preceq$. From $\theta' x \preceq \tau' x$ and the impossibility of replacing the word τ' with the word θ' in σ by transformations $OPT - LB(\theta' x(\tau')^{-1}, \sigma)$, $OPT - RB(\tau' x(\theta')^{-1}, \sigma)$ with a possible transformation $REM - R$, it follows that $\sigma' = \theta'$, i.e., $\sigma = \tau' x\theta'$. Since the transformation $ORD(\sigma)$ is impossible, and by assumption $\theta' \preceq \tau'$, then $|\theta'| < |\tau'| = k$. Therefore, $|\sigma| < k + 1 + k = 2k + 1$, which contradicts $|\sigma| \ge 2k + 1$. The resulting contradiction proves the falsity of the assumption about the inapplicability of the lemma for $i = k + 1$ under the condition of case (a). The falsity of the assumption that the Lemma is not valid for the case (b) is

proven dual to case (a) and for case (c) – similarly to case (a). □

Next, we will show that $< C, L >= (C_G, L_G)$. Let $\sigma \in < C >$. From the previous performing of the operation $ORD(\sigma)$, it follows that $\sigma \preceq \sigma^{-1}$ and $\sigma^{-1} \notin < C >$. From Lemma 4, it follows that σ is a canonical word for G, and in the case of oddness $|\sigma|$, $\sigma^{-1} \notin < C >$. Therefore, taking into account Lemma 3, $\sigma \in C_G$, i.e., $< C > \subseteq C_G$.

Let's show that $C_G \subseteq < C >$. Let $\sigma \in C_G$, vertices v', v'' are generators for σ, $\xi(v') = x'$, $\xi(v'') = x''$. For definiteness, let $\sigma = sp_{v'}(sp_{v''})^{-1}$. By the AC-algorithm, $\sigma \preceq \sigma^{-1}$, i.e., $sp_{v'}(sp_{v''})^{-1} \preceq sp_{v''}(sp_{v'})^{-1}$. Since $< C, L >$ is a defining pair for $\tilde{G}$, then according to the AP-algorithm, there must exist a word $\tau \in < C > \bigcup < L >$ such that the edge $\{v', v''\}$ is included in the walk $p_0(\tau)$. Since, according to the AC-algorithm, $sp_{v'} \neq sp_{v''}x'$ and $sp_{v''} \neq sp_{v'}x''$, it follows from Lemma 4 that $\tau \notin < L >$, and, depending on the parity of the number $|\sigma|$, either $\tau = sp_{v'}(sp_{v''})^{-1}$ or $\tau = sp_{v''}(sp_{v'})^{-1}$. From the previous performing of the operation $ORD(\sigma)$, considering that $sp_{v'}(sp_{v''})^{-1} \preceq sp_{v''}(sp_{v'})^{-1}$, it follows that $\tau = sp_{v'}(sp_{v''})^{-1}$, i.e. $\tau = \sigma$, so $C_G \subseteq < C >$.

We will show that $< L >= L_G$. Since $< C, L >$ is a defining pair for $\tilde{G}$, for every word $\rho \in < L >$, the vertex $v_0\rho$ is leaf ($v_0\rho \neq v_0$), and for every leaf vertex v of the graph $\tilde{G}$ other than v_0, there exists at least one word $\rho' \in < L >$ such that $v_0\rho' = v$. Taking into account the previous performing of the operation $DEL - REP(\tau)$ and Lemma 2, we obtain that $< L >$ consists of all such words $\tau \in \mathcal{V}_G$ that $v_0\tau \neq v_0$, and the vertex $v_0\tau$ is leaf in the graph $\tilde{G}$, therefore, according to the AC-algorithm, $< L >= L_G$. We obtained $< C, L >= (C_G, L_G)$. □

Thus, the procedure for checking whether a given correct pair (C, L) is a defining pair for a given graph $\tilde{G}$ boils down to performing a compression (C, L), constructing the canonical defining pair for $\tilde{G}$, and comparing $< C, L >$ with (C_G, L_G): if $< C, L >= (C_G, L_G)$, then (C, L) is a defining pair for $\tilde{G}$; if $< C, L > \neq (C_G, L_G)$, then (C, L) is not a defining pair for $\tilde{G}$. In our opinion, in computational terms, the proposed method is simpler than constructing the graph $G(C, L)$, and then comparing $G(C, L)$ with $\tilde{G}$. In particular, the pair (C, L) from Figure 4, is a defining pair for the graph $\tilde{G}$ from Figure 2, since $< C, L >= (C_G, L_G)$.

It is easy to see that applying compression to the canonical defining pair for the D-graph $\tilde{G}$ will not change it. Therefore, taking into account Theorem 3, the canonical defining pair (C_G, L_G) is the smallest in terms of power and volume of each component defining pair for the graph $\tilde{G}$.

The transformations proposed in this and the previous sections can form the basis for effective solutions to various applied problems related to deterministic graphs. The solution to one such problem, namely, finding the shortest paths from the basepoint vertex to all other vertices of a deterministic graph, is currently in processing. However, some questions remain open: assessment of the complexity of the compression algorithm, the possibility of adapting the compression algorithm for cases where the correctness of the source pair is unknown in advance.

Based on the algorithms presented in this article, a prototype software implementation [16] was developed in the form of a library of functions that extend the capabilities of the NetworkX package. Using this library, it is possible to construct a graph by a given pair, the canonical defining pair of a D-graph, perform compression of a given correct pair, and compare the result of compression with the canonical defining pair of a given graph. We also investigated some test metrics of this program implementation. When implementing the $PATH - OPT(C, L)$ operation, among its possible variants, the one with the best test metric was chosen. The library includes a description of our program implementation.

Conclusion

The article proposes the compression algorithm that, if the source pair is correct, transforms it into a canonical defining pair for a D-graph for which the source pair is a defining pair. A prototype software implementation of this algorithm has been developed. A number of open problems related to the linguistic presentation of D-graphs and the compression algorithm are outlined. Also, in our opinion, the presented compression algorithm can be the basis for solving a number of applied problems related to the linguistic presentation of D-graphs by a defining pair.

References

[1] Kramer, F., Kramer, H.: Ein Färbungsproblem der Knotenpunkte eines Graphen bezüglich der Distanz p. Rev. Roum. Math. Pures et Appl. **14**(7), 1031–1038 (1969). https://doi.org/10.13140/RG.2.2.24729.83047

[2] Hahn, G., Kratochvíl, J., Širáň, J., Sotteau, D.: On the injective chromatic number of graphs. Discrete Math. **256**(1–2), 179–192 (2002). https://doi.org/10.1016/S0012-365X(01)00466-6

[3] Bertossi, A.A., Bertossi, M.A.: Code assignment for hidden terminal interference avoidance in multihop packet radio networks. IEEE/ACM Transactions on Networking **3**(4), 441–449 (1995). https://doi.org/10.1109/90.413218

[4] Dudek, G., Jenkin, M.: Computational Principles of Mobile Robotics, 2nd ed. Cambridge Univ. Press (2010). https://doi.org/10.1017/CBO9780511780929
[5] Baier, C., Katoen, J.-P.: Principle of Model Checking. MIT Press (2008)
[6] Stepkin, A.V.: Using a collective of agents for exploration of undirected graphs. Cybern. Syst. Anal. **51**, 223–233 (2015). https://doi.org/10.1007/s10559-015-9715-z
[7] Sapunov, S.V.: Directional movement of a collective of compassless automata on a square lattice of width 2. Cybern. Syst. Anal. **60**, 899–906 (2024). https://doi.org/10.1007/s10559-024-00727-x
[8] Senchenko, O.S., Prytula, M.I.: The presentation of deterministic and strongly deterministic graphs. Algebra Discrete Math. **36**(2), 225–242 (2023). https://doi.org/10.12958/adm2157
[9] Grunskii, I., Mikhaylova, I., Sapunov, S.: Domination on the vertices of labeled graphs. Algebra Discrete Math. **14**(2), 174–184 (2012)
[10] Vasilevskii, M.P.: Failure diagnosis of automata. Cybern. Syst. Anal. **9**, 653–665 (1973). https://doi.org/10.1007/BF01068590
[11] Chow, T.S.: Testing software design modeled by finite-state machines. IEEE Transactions on Software Engineering **SE-4**(3), 178–187 (1978). https://doi.org/10.1109/TSE.1978.231496
[12] Zhuchok, Yu.V.: The monoid of endomorphisms of disconnected hypergraphs. Algebra Discrete Math. **16**(1), 134–150 (2013)
[13] Zhuchok, A.V.: Certain verbal congruences on the free trioid. Algebra Discrete Math. **37**(2), 287–303 (2023). https://doi.org/10.12958/adm2173
[14] Zhuchok, A.V., Zhuchok, Yul.V., Zhuchok, Yu.V.: Certain congruences on free trioids. Comm. Algebra **47**(12), 5471–5481 (2019). https://doi.org/10.1080/00927872.2019.1631322
[15] Grunskii, I.S., Senchenko, A.S.: Properties of systems of defining relations for automata. Discrete Math. Appl. **14**(6), 593–601 (2004). https://doi.org/10.1515/1569392043272458
[16] NickIT87: IAMM_proceedings: Linguistic representation of deterministic graphs. GitHub repository. https://github.com/NickIT87/IAMM_proceedings. Accessed 17 August 2026

CONTACT INFORMATION

O. S. Senchenko, M. I. Prytula
Institute of Applied Mathematics and Mechanics of the National Academy of Sciences of Ukraine
E-Mail: Senchenko.a76@gmail.com, elanir358@gmail.com

Received by the editors: 03.02.2025
and in final form 22.07.2026.