SN Computer Science (2022) 3:113
https://doi.org/10.1007/s42979-021-01001-z

ORIGINAL RESEARCH

Some Consistency Rules for Graph Matching

Badreddine Benreguia[1] · Hamouma Moumen[1]

Received: 11 February 2021 / Accepted: 17 December 2021 / Published online: 29 December 2021

Abstract
Graph matching is a comparison process of two objects represented as graphs through finding a correspondence between vertices and edges. This process allows defining a similarity degree (or dissimilarity) between the graphs. Generally, graph matching is used for extracting, finding and retrieving any information or sub-information that can be represented by graphs. In this paper, a new consistency rule is proposed to tackle with various problems of graph matching. After, using the proposed rule as a necessary and sufficient condition for the graph isomorphism, we generalize it for subgraph isomorphism, homomorphism and for an example of inexact graph matching. To determine whether there is a matching or not, a backtracking algorithm called CRGI2 is presented who checks the consistency rule by exploring the overall search space. The tree-search is consolidated with a tree pruning technique that eliminates the unfruitful branches as early as possible. Experimental results show that our algorithm is efficient and applicable for a real case application in the information retrieval field. On the efficiency side, due to the ability of the proposed rule to eliminate as early as possible the incorrect solutions, our algorithm outperforms the existing algorithms in the literature. For the application side, the algorithm has been successfully tested for querying a real dataset that contains a large set of e-mail messages.

Keywords Graph matching · Isomorphism · Consistency rule · Information retrieval · Tree-search

Introduction

Graphs are one of the most general forms of data representation. They are able to represent structural relations that could exist between different parts of an object. This crucial benefit led to the growing use of graphs in many fields including image processing, information retrieval, bioinformatics and networking. The benefit is not only in the graph itself for data representation, but in performing other operations based on graphs like graph matching problem. The graph matching is the process of comparing two graphs that aims to define a similarity degree between the graphs. Two main categories of algorithms can be distinguished. The first includes exact matching algorithms that require a strict correspondence among two graphs or at least among their subgraphs. The second category contains inexact matching algorithms where a matching can occur even if both compared graphs are structurally different. In its form, the most rigorous, the exact graph matching is known as graph isomorphism in which a one-to-one correspondence must be found between each vertex of the first graph and each vertex of the second graph, such that the edge connections are respected. Other forms of the exact graph matching are the subgraph isomorphism, the most common subgraph of two graphs and the homomorphism. In the case of the subgraph isomorphism problem, the goal is to know if there is an isomorphism between the first graph and a subgraph of the second graph.

In practical applications, the inexact graph matching problem has been intensively treated in the literature [1–4]. Most of the existing methods formulate the graph matching as an optimization problem. A cost function must find a correspondence of vertex-to-vertex and edge-to-edge, such that the cost is minimized to enhance the similarity between graphs.

It is well known that the problem of subgraph isomorphism is NP-complete. However, up to now, it is unknown whether the graph isomorphism is an NP-complete problem or a polynomial problem [5–7]. Polynomial isomorphism algorithms have been developed for particular kinds of

Hamouma Moumen
hamouma.moumen@univ-batna2.dz

Badreddine Benreguia
badreddine.benreguia@univ-batna2.dz

1 University of Batna 2, 05000 Batna, Algeria

graphs (trees, planar graphs, bounded valence graphs) but no polynomial algorithms are known for the general case.

Whatever the matching problem is exact or not, the goal is to find the correspondence (or the permutation) to define the similarity. Generally, the problem is tackled in literature using permutation theory. To find the optimal permutation (or subpermutation) of the matching, there are several techniques of exploration, the most studied are: exhaustive tree-searching, continuous optimization and heuristics. Usually, in such context, the goal is to determine the solution as early as possible, in particular when the search space grows. In real applications, like information retrieval using big data, it will be interesting to have new developments on the graph matching problem to accelerate exploration with efficient results.

Paper contribution

Algorithms based on spectral methods use the famous equality $M' = P^tMP$, where M and M' represent the adjacency matrices of the two graphs. Whereas P is the unknown permutation that we have to define. The equation shows that graphs of M and M' are isomorphic, if there is at least a permutation matrix P that could solve $M' = P^tMP$.

Using this equation, Ullmann [8] has illustrated that is possible to show that two graphs are isomorphic if there is some permutation p such that: for each element $m_{i,j}$ of the matrix M, if $m_{i,j} = 1$, there is a corresponding element in M' where $m'_{p(i),p(j)} = 1$.

In this paper, we give a simple general equality (called consistency rule) using matrix elements of the graphs. For two isomorphic graphs, we prove for all elements, $[m_{i,j}] = [m'_{p(i),p(j)}]$ which is more general than the rule of Ullmann. The new proposed consistency rule is a necessary and sufficient condition for graph isomorphism, in contrast to Ullmann's rule which is only a necessary condition. We show also that the proposed rule could be used for various problems in graph matching using any other square matrix representation like Laplacian matrix and distance matrix.

The rest of the paper is organized as follows. Previous works in relation to this paper are discussed in Section "Related Work". In Section "Definitions", basic definitions and some theoretical aspects are given. The main idea of the proposed consistency rule for the graph isomorphism problem is presented in Section "Consistency Rule for Graph Isomorphism Problem". An algorithm of graph isomorphism based on tree-search exploration with backtracking and branch pruning techniques is introduced in Section "Algorithm CRGI2 for Graph Isomorphism". In Sections Subgraph Isomorphism and Homomorphism, An Inexact Matching for Attributed Graphs and Tree Matching for XML Retrieval, the proposed consistency rule is generalized for subgraph isomorphism, homomorphism, inexact graph matching and has been suited for the problem of XML retrieval. Experimental tests are carried out, in Section "Experimental Results", to show the performance and the applicability of CRGI2. Final notes and conclusions are given in Section "Conclusion".

Related Work

Graph isomorphism algorithms use mainly two approaches: the direct approach and the indirect approach. In the direct approach, the algorithms try to find an isomorphism between the two input graphs directly with a backtracking algorithm using feasibility rules to prune the tree-search. Ullmann's algorithm, Schmidt and Druffel's algorithm (called SD), VF, VF2, and CRGI represent a collection of this approach [7–10]. Note that most of isomorphism algorithms of this approach are used too for subgraph isomorphism like Ullmann, VF, and VF2.

Nauty algorithm [11] is the main algorithm of the indirect approach in which canonical labeling of each graph must be calculated. Two graphs can be checked for isomorphism by simply verifying the equality of the adjacency matrices of their canonical forms. An interesting algorithm that combines between Nauty and VF2 is presented in [5]. Other existing techniques, such as non-deterministic ones (stochastic), are so powerful to reduce the complexity from exponential to polynomial, but they are not guaranteed to find an exact and optimal solution.

For the inexact graph matching problem, the most known paradigms are a graph edit distance [12–14], b graph kernels and embedding [15–18], **c** spectral algorithms [3, 4, 19, 20] and **d** algorithms based on deep learning [21, 22]. The reader can refer to the surveys [23–26] for more details on graph matching algorithms classification.

Graph edit distance (GED), represents the main way for the inexact graph matching process. Most of the algorithms, in this case, uses a function *cost* that quantifies the errors between the two graphs. Graph edit distance, which is recognized as one of the most flexible and universal error-tolerant matching paradigms, is based on computing the cost of the needed operations (vertex insertion, vertex deletion, etc.) to transform one graph to obtain the second graph. At a lower cost (the number of operations is reduced), there will be more similarity between graphs. The survey [27] gives more details of algorithms based on the edit distance paradigm. Note that in [1], the most common subgraph problem has been proved that it is a special case of graph edit distance computation. This later shows a form of combination between exact and inexact graph matching.

Spectral matching methods are based on the fact that the eigenvalues of a matrix remain unchanged whatever the rows and columns are permuted. The general idea is based on using matrices of the corresponding graphs. The eigenvalues and the eigenvectors of the adjacency or Laplacian matrix of a graph are invariant with respect to vertex permutation. Hence, if two graphs are isomorphic, their structural matrices will have the same eigendecomposition. By representing the underlying graphs by means of the eigendecomposition of their structural matrix, the matching process of the graphs can be conducted on some features derived from their eigendecomposition.

Definitions

Definition 1 A *graph* $G = (V, E)$ consists of a finite non-empty set $V = \{0, 1, ..., n-1\}$ of n vertices and a set of edges $E \subseteq V \times V$.

Definition 2 Let $G = (V, E)$. The *adjacency matrix* of G is $n \times n$ matrix M defined as follows:

$$M = [m_{ij}]\begin{cases} m_{ij} = 1 & \text{if } (i,j) \in E \\ m_{ij} = 0 & \text{otherwise} \end{cases}$$

Definition 3 Let $G = (V, E)$. The *distance matrix* $D = [d_{ij}]$ is $n \times n$ matrix in which the element d_{ij} represents the length of the short path between the vertices i and j. If $i = j$, then $d_{ij} = 0$. If no path exists between the two vertices, the length is defined to be infinite.

Definition 4 A *permutation* p on a set V is a bijection $p : V \to V$. If $V = \{0, 1, ..., n-1\}$, a permutation of V is called a *permutation on n vertices*.

Example of a permutation:

$$p = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 3 & 6 & 0 & 5 & 1 & 2 \end{pmatrix}$$

which means $p(0) = 4, p(1) = 3, p(2) = 6, ..., p(6) = 2$.

Definition 5 A *subpermutation* p on a set V is an injection $p : S \to V$, where $S \subset V$.

Example of a subpermutation on 4 vertices, where $V = \{0,1,...,6\}$:

$$p = \begin{pmatrix} 0 & 2 & 3 & 5 \\ 4 & 6 & 0 & 1 \end{pmatrix}$$

Definition 6 Let $G = (V, E)$ and $G' = (V, E')$ be two graphs. G and G' are called *isomorphic* if there exists a permutation $p : V \to V$ such that: $(i,j) \in E$ if and only if $(p(i), p(j)) \in E'$.

Definition 7 Given two graphs $G = (V, E)$ and $G' = (V', E')$, where $V \subset V'$. There is a *subgraph isomorphism* from G to G' if there exists a subpermutation $p : V \to V'$ such that: $(i,j) \in E$ if and only if $(p(i), p(j)) \in E' \cap (S \times S)$, where $S \subset E'$ and $|S| = |E|$.

Definition 8 Let $G = (V, E)$ and $G' = (V, E')$ be two graphs. There is a graph *homomorphism* from G to G', if there exists a permutation $p : V \to V$ such that: $(i,j) \in E \Rightarrow (p(i), p(j)) \in E'$.

Definition 9 Let p be a permutation on n vertices. The *permutation matrix* P is an $n \times n$ matrix defined as follows:

$$P = [p_{ij}]\begin{cases} p_{ij} = 1 & \text{if } p(j) = i \\ p_{ij} = 0 & \text{if not} \end{cases}$$

For example, the permutation matrix P of the permutation p shown in Definition 4 is:

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

Actually, the permutation matrix is obtained from the identity matrix by permuting the columns (i.e. switching some of the columns).

Definition 10 A *graph invariant* is a function f applied on isomorphic graphs such that, if G and G' are isomorphic, then $f(G) = f(G')$ (the converse is not necessarily true: $f(G) = f(G')$ does not mean G and G' are isomorphic). Therefore, an invariant imposes a necessary condition for isomorphism.

Simple graph invariants are the number of vertices and the number of edges of a graph. Other graph invariants are the determinant, the characteristic equation of its adjacency matrix, and the set of its roots (the spectrum of the graph).

The vertex invariant is another invariant which is extended to graph invariant. *Node invariant* is a function f on a node, such that if there is an isomorphism p between G and G', for each $v \in G$, then $f(v) = f(p(v))$.

Proposition 1 *Let M and M' be the adjacency matrices of G and G', respectively. The graphs G and G' are isomorphic if and only if there exists a permutation matrix P such that:*

$$M' = P^t M P \tag{1}$$

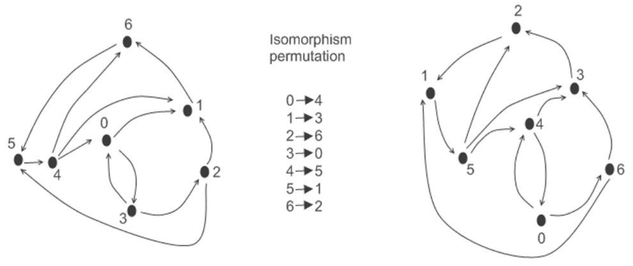

Fig. 1 Example of two isomorphic graphs with the corresponding permutation

Proof Given isomorphic graphs, the isomorphism gives a permutation of the vertices, which leads to a permutation matrix. Similarly, the permutation matrix gives an isomorphism. As for the equality, it is explained by an example in Section "Example of Isomorphic Graphs".

Consistency Rule for Graph Isomorphism Problem

Equation $M' = P^tMP$ has been widely used in the literature to solve the problem of graph isomorphism [4, 6, 8]. It is clear that M and M' are given matrices of the graphs G and G' while P is the unknown permutation must be determined. If Eq. (1) has no solution, i.e. P does not exist, no isomorphism could exist.

Until now, there is no way to solve (1) in polynomial time. To determine P (or in-existence of P), an exhaustive search must be executed in the area of all possible permutations (Brute Force search). Ullmann's algorithm [8] gives an example of this process.

The disadvantage of this method is the high complexity given in $O(n!)$. To reduce the computing complexity, Eq. (1) will be reformulated. It will be interesting to give an example to understand the behavior of (1).

Example of Isomorphic Graphs

Let $G = (V, E)$ and $G' = (V, E')$ be the isomorphic graphs shown in Fig. 1.

The corresponding adjacency matrices of G and G' are, respectively:

$$M = \begin{pmatrix} 0&1&0&1&0&0&0\\ 0&0&0&0&0&0&1\\ 0&1&0&0&0&1&0\\ 1&0&1&0&0&0&0\\ 1&1&0&0&0&0&1\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0 \end{pmatrix},$$

$$M' = \begin{pmatrix} 0&0&0&0&1&0&1\\ 0&0&0&0&0&1&0\\ 0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0\\ 1&0&0&1&0&0&0\\ 0&0&1&1&1&0&0\\ 0&1&0&1&0&0&0 \end{pmatrix}$$

The permutation matrix (which is known) is:

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

Using (1), the first matrix multiplication P^tM is:

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

$$= \begin{pmatrix} 1&0&1&0&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1\\ 0&1&0&1&0&0&0\\ 1&1&0&0&0&0&1\\ 0&1&0&0&0&1&0 \end{pmatrix} = M''$$

Using (1), the second matrix multiplication $M''P$ i.e. P^tMP is:

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

$$= \begin{pmatrix} 0&0&0&0&1&0&1 \\ 0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0 \\ 1&0&0&1&0&0&0 \\ 0&0&1&1&1&0&0 \\ 0&1&0&1&0&0&0 \end{pmatrix} = M'$$

A depth observation of the above matrix multiplications allows concluding that:

- The multiplication $M'' = P^t M$ allows to each row i of M changing its position i into the new position $p(i)$ in M''. Thus,

  $$M'' = P^t M = R_p(M), \tag{2}$$

  where, R_p is a *position transformation* function that transforms each row i to $p(i)$ i.e.

  $$\left[m_{i,j}\right] \longrightarrow_{R_p} \left[m''_{p(i),j}\right]. \tag{3}$$

  If we select any row from M, for example row 3 will be appeared at the position 0 in M''.
- The second multiplication $M' = M''P$ allows to each column j of M'' changing its position j into the new position $p(j)$ in M'. Thus,

  $$M' = M''P = C_p\left(M''\right), \tag{4}$$

  where, C_p is a *position transformation* function that transforms each column j to $p(j)$ i.e.

  $$\left[m''_{i,j}\right] \longrightarrow_{C_p} \left[m'_{i,p(j)}\right]. \tag{5}$$

  If we select any column from M'', for example column 5 will be appeared at the position 1 in M'.

In general, the position transformations (3) and (5) are still correct even for adjacency matrices of attributed graphs defined above. The fact that the permutation matrix is gotten from the identity matrix by permuting the columns (or rows), will lead to conserve the values of the cases but changing their positions.

By replacing (2) in (4):

$$M' = C_p\left(R_p(M)\right). \tag{6}$$

Therefore, M' is obtained by a transformation of rows positions under p in first step, and in second step, a transformation of columns positions under the same permutation p.[1]

Now, we apply transformations R_p and C_p on $[m_{i,j}]$ sequentially. Using (3) and (5), we get:

$$\left[m_{i,j}\right] \longrightarrow_{R_p} \left[m''_{p(i),j}\right] \longrightarrow_{C_p} \left[m'_{p(i),p(j)}\right]. \tag{7}$$

As mentioned, R_p and C_p are functions for changing positions that preserve values, in other words, R_p and C_p do not influence the cases contents after position transformation, this means from (7) that for each case $m_{i,j}$ of M there exists a corresponding case in M'' and another in M' such that:

$$m_{i,j} = m''_{p(i),j} = m'_{p(i),p(j)}. \tag{8}$$

If we select any case $m_{i,j}$ from M, for example $m_{2,1}$ will be appeared at the position (6, 1) in M'', and then at the position (6, 3) in M' with value preservation, i.e. $m_{2,1} = m''_{6,1} = m'_{6,3} = 1$.

To obtain M' from M, each case of M is a subject of double position transformation (changing the row i and then changing the column j), this position transformation must be done under a given permutation p.

From above, we have this proposition which is valid also for attributed graphs:

Proposition 2 *Let $M = [m_{i,j}]$ and $M' = [m'_{i,j}]$ be the adjacency matrices of G and G′ (both are simple or both are attributed), respectively. The graphs G and G′ are isomorphic if and only if there exists a permutation $p : V \to V$ such that*:

$$\forall i,j \qquad m_{i,j} = m'_{p(i),p(j)}. \tag{9}$$

Besides the adjacency matrix, Proposition (1) may be extended to other graph representations like Laplacian matrix, distance matrix and even for matrices of attributed graphs. Thus, from the previous explained notion of cases transformation, we have theses propositions too:

Proposition 3 *Let X and X′ be square matrices*[2] *of G and G′, respectively. The graphs G and G′ are isomorphic if and only if there exists a permutation matrix P such that*:

$$X' = P^t X P \tag{10}$$

The later equation has been used in literature with only adjacency matrix [4, 6, 8]. However, if we consider using another representation like the distance matrix for the previous graphs of Fig. 1.

[1] Observe that $M' = R_p(C_p(M))$ is also true.

[2] Several square matrices that contain informations about the structure of the graph can be formed. In this case, rows and columns are indexed by the vertices of the graph. Some of the most commonly studied square matrices for representing graphs are the adjacency matrix, the distance matrix and the Laplacian matrix.

$$D = \begin{pmatrix} 0&1&2&1&4&3&2\\ 4&0&6&5&3&2&1\\ 3&1&0&4&2&1&2\\ 1&2&1&0&3&2&3\\ 1&1&3&2&0&2&1\\ 2&2&4&3&1&0&2\\ 3&3&5&4&2&1&0 \end{pmatrix},$$

$$D' = \begin{pmatrix} 0&2&3&2&1&3&1\\ 3&0&2&2&2&1&4\\ 4&1&0&3&3&2&5\\ 5&2&1&0&4&3&6\\ 1&3&2&1&0&4&2\\ 2&2&1&1&1&0&3\\ 4&1&2&1&3&2&0 \end{pmatrix}$$

Using (10) such that X is the distance matrix D, we have $P^tDP =$

$$\begin{pmatrix} 0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1\\ 0&1&0&0&0&0&0\\ 1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&1&0&0&0&0 \end{pmatrix} \times \begin{pmatrix} 0&1&2&1&4&3&2\\ 4&0&6&5&3&2&1\\ 3&1&0&4&2&1&2\\ 1&2&1&0&3&2&3\\ 1&1&3&2&0&2&1\\ 2&2&4&3&1&0&2\\ 3&3&5&4&2&1&0 \end{pmatrix} \times \begin{pmatrix} 0&0&0&0&1&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1\\ 1&0&0&0&0&0&0\\ 0&0&0&0&0&1&0\\ 0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0 \end{pmatrix}$$

$$= \begin{pmatrix} 1&2&1&0&3&2&3\\ 2&2&4&3&1&0&2\\ 3&3&5&4&2&1&0\\ 4&0&6&5&3&2&1\\ 0&1&2&1&4&3&2\\ 1&1&3&2&0&2&1\\ 3&1&0&4&2&1&2 \end{pmatrix} \times \begin{pmatrix} 0&0&0&0&1&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1\\ 1&0&0&0&0&0&0\\ 0&0&0&0&0&1&0\\ 0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0 \end{pmatrix}$$

$$= \begin{pmatrix} 0&2&3&2&1&3&1\\ 3&0&2&2&2&1&4\\ 4&1&0&3&3&2&5\\ 5&2&1&0&4&3&6\\ 1&3&2&1&0&4&2\\ 2&2&1&1&1&0&3\\ 4&1&2&1&3&2&0 \end{pmatrix} = D'$$

Proposition 4 *Let $X = [x_{i,j}]$ and $X' = [x'_{i,j}]$ be square matrices of $G = (V, E)$ and $G' = (V, E')$, respectively. The graphs G and G' are isomorphic if and only if there exists a permutation $p : V \rightarrow V$ such that*:

$$\forall i,j \qquad x_{i,j} = x'_{p(i),p(j)}. \tag{11}$$

This proposition is a consistency rule that can be used also for matching weighted graphs in which the contents of the adjacency matrices are real. In this case, the rule will be formulated as $|x_{i,j} - x'_{p(i),p(j)}| \leq \epsilon$, where ϵ is a threshold that can define the quality of graph matching. It is clear that the problem turns back to graph isomorphism when $\epsilon = 0$.

This later proposition will be used as consistency rule (necessary and sufficient condition) for testing graph isomorphism in the proposed algorithm. It will be generalized, after, for other problems of graph matching. Therefore, its interest is that allows an early elimination of invalid permutations. When there exists at least one case (i, j) such that $x_{i,j} \neq x'_{p(i),p(j)}$, this means that the permutation p could not be an isomorphism, and any other permutation p' must be tested. If all permutations do not satisfy (11), no isomorphism can exist.

Previously, we assumed working with directed graphs. In fact, rules 9 and 11 can also be applied to prove the graph isomorphism on this type of graphs:

- *Undircted graphs* A graph is undirected if edges have no direction. If there is an edge from i to j in an undirected graph, then there is also an edge from j to i. Adjacency matrix X of undirected graph is always symmetric where $x_{i,j} = x_{j,i}$. Therefore, it is sufficient to prove the graph isomorphism by using only the upper triangular part of the adjacency matrix where $j \geq i$ (or only the lower triangular part where $i \geq j$). Thus, undirected graphs G and G' (represented by symmetric matrices X and X') are isomorphic if and only if there exists a permutation $p : V \rightarrow V$ such that:

$$\forall i,j : j \geq i \qquad x_{i,j} = x'_{p(i),p(j)}. \tag{12}$$

- *Graphs with loops* A loop (or self-edge) is an edge (i, i) from a vertex i to itself. Loops correspond to a diagonal entry in the adjacency matrix M of the graph. If there is a loop (i, i), then $m_{i,i} = 1$, otherwise $m_{i,i} = 0$. Observe that consistency rules 9 and 11 remain true for both cases (1) $i = j$ (loops) or even (2) $i \neq j$.
- *Multigraphs* In some cases, there can be more than one edge between the same pair of vertices. Those edges are known as multi-edges. A graph with multi-edges is called a *multigraph*. A multi-edge is represented by setting the corresponding matrix element $m_{i,j}$ equal to the multiplicity of the edge. For example, if there 3 edges between vertices i and j, then $m_{i,j} = 3$. Obviously, using adjacency matrices by representing edge multiplicity, rules 9 and 11 remain true.

Note that a graph that has neither self-edges (loops) nor multi-edges is called a *simple* graph. In the remainder of this paper, we assume working with simple graphs, unless the opposite is indicated.

```
(1)  Algorithm
(2)  INPUT A square matrices $X$ and $X'$ of the same size $n$
(3)  OUTPUT An isomorphism $p$ or no-isomorphism
(4)  BEGIN
(5)      Define the initial search space $S_0 \times S_1 \times ... \times S_{n-1}$
(6)      Compute partitions of $X$ and $X'$ by vertex invariant
(7)      IF partitions of $X$ and $X'$ are not compatible THEN
(8)          Return no-isomorphism
(9)      ENDIF
(10)     FOREACH vertex $i \in V$
(11)         Reduce $S_i$ using partitions of $X$ and $X'$
(12)     ENDFOR
(13)     REPEAT
(14)         Select a novel branch $p = (p(0), ..., p(n-1))$
(15)         IF consistency rule (11) is satisfied THEN
(16)             Return $(p)$ and terminate the algorithm
(17)         ENDIF
(18)     UNTIL end of tree
(19)     Return no-isomorphism
(20) END Algorithm CRGI2
```

Fig. 2 Algorithm for graph isomorphism—CRGI2

For the subgraph isomorphism problem, we introduce the following proposition:

Proposition 5 *Let* $X = [x_{i,j}]$ *and* $X' = [x'_{i,j}]$ *be square matrix representations of* $G = (V, E)$ *and* $G' = (V', E')$, *respectively, where* $|V| < |V'|$. *There exists a subgraph isomorphism from* G *to* G' *if and only if there exists a subpermutation* $p : V \to V'$ *such that*:

$$\forall i, j \in V \qquad x_{i,j} = x'_{p(i),p(j)}. \tag{13}$$

It is clear that in the subgraph isomorphism problem there is a graph smaller than the other. In this case, we are interesting to find for each case of the smaller matrix an associated case of the wider matrix.

Algorithm CRGI2 for Graph Isomorphism

Suppose we choose a square matrix as graph representation like adjacency matrix or distance matrix or Laplacian matrix. A description of CRGI2[3] algorithm is written as shown in Fig. 2.

Perhaps, the most natural way to tackle the graph isomorphism problem is using direct backtracking. After partitioning vertices using a selected invariant to reduce the search space, the algorithm must explore the possible permutations of vertices. If a branch (i.e. a permutation) does not reach a valid solution according to (11), the algorithm backtracks to check another permutation.

The exploration is expressed in the algorithm by the iteration *Select a novel branch* $(p(0), ..., p(n-1))$ without presenting details of the implementation. However, before exploring the tree-search, vertices partitions are computed first to check their compatibility. If partitions are not compatible, the graphs cannot be isomorphic. Otherwise, the algorithm computes the reduced search space derived from the partitions to find a prospective isomorphism.

The Initial Search Space

In fact, it is hard to explore the whole search space to find the isomorphism permutation. CRGI2 uses directly a reduceable search space deducted from vertices partitions. In this section, the theoretical search space is discussed to show how it can be reduced by vertices invariant.

The possible correspondences for each vertex i, are:

- The first vertex 0 of G can be associated to any vertex of G', thus: $p(0) \in \{0, 1, 2, ..., n-1\} = S_0$
- Once the vertex $p(0)$ of G' has been allocated to the vertex 0 of G, the second vertex 1 of G can be associated to any vertex of G' except $p(0)$, thus: $p(1) \in S_0 - \{p(0)\} = S_1$
- Once $p(0), ..., p(i-1)$ of G' have been allocated to vertices $0, ..., i-1$ of G, respectively. Possible correspondences of the vertex i are: $p(i) \in S_0 - \{p(0), ..., p(i-1)\} = S_i$
- Once $p(0), ..., p(n-2)$ have been allocated, the possible correspondences of the vertex $n-1$ are: $p(n-1) \in S_0 - \{p(0), ..., p(n-2)\} = S_{n-1}$

An isomorphism is a permutation $p = (p(0), ..., p(n-1)) \in S_0 \times S_1 \times ... \times S_{n-1}$ that satisfies (11). Obviously, the size of the search space $S_0 \times S_1 \times ... \times S_{n-1}$ is $n!$. Therefore, the complexity is $O(n!)$

Reducing the Search Space by Vertex Invariant

Computing the partitions of vertices using a vertex invariant is an important step that allows reducing the search space. Vertex invariants like degree, vector of distances (the number of vertices reachable at each distance) or others can be selected for vertices partition. Using the out-degree as invariant in our example, the possible correspondences of vertex 0 are $\{0, 4, 6\}$. Thus, vertices partition using out-degree gives:

Partition of vertices in G is $\{\{1, 5, 6\}, \{0, 2, 3\}, \{4\}\}$

Partition of vertices in G' is $\{\{1, 2, 3\}, \{0, 4, 6\}, \{5\}\}$ which means that the elements of the first subset $\{1, 5, 6\}$ of G can be associated solely to the elements of the first subset

[3] To denote: Consistency Rule for Graph Isomorphism. We call CRGI2 to indicate the updated version of CRGI.

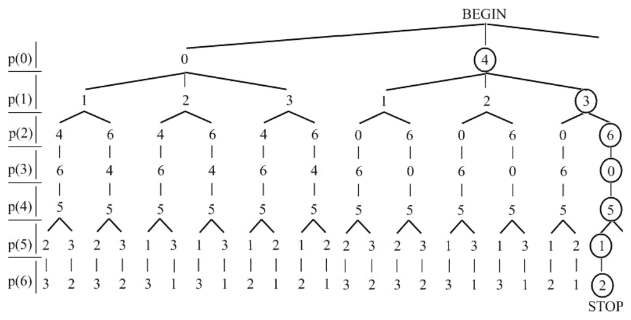

Fig. 3 Tree-search exploration without branch pruning

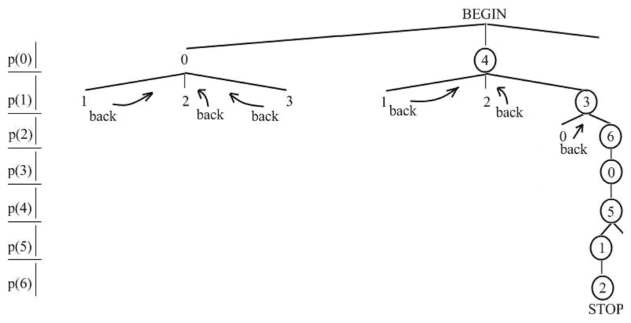

Fig. 4 Tree-search exploration with branch pruning

$\{1, 2, 3\}$ of G' and so on for the rest subsets. In this case, we know a sub-solution i.e. 4 is associated to 5.

Vertices partition allows to construct the search space $S_0 \times S_1 \times ... \times S_{n-1}$ which is more reduced than initially. Thus, $S_0 = \{0, 4, 6\}$

$S_1 = \{1, 2, 3\}$

$S_2 = \{0, 4, 6\} - \{p(0)\}$

$S_3 = \{0, 4, 6\} - \{p(0), p(2)\}$

$S_4 = \{5\}$

$S_5 = \{1, 2, 3\} - \{p(1)\}$

$S_6 = \{1, 2, 3\} - \{p(1), p(5)\}$

The tree-search in Fig. 3 shows all the explored branches using the reduced search space until an isomorphism is found.

More Reduction During Tree Exploration

The last step performed by CRGI2 is the exploration of the tree-search. To enhance the efficiency of the algorithm, CRGI2 is able to prune the tree-search during the exploration. At each depth l of the tree, unsuccessful branches are discarded as soon as possible. The consistency rule (11) is checked at least for the subpermutation $(p(0), ..., p(l-1))$. If there is a prospective case $x_{i,j}$ such that $x'_{p(i),p(j)} \neq x_{i,j}$, an early pruning of unnecessary branch will be done as shown in Fig. 4. If the rule is still valid at least for the subpermutation, the exploration continues for more depth. Otherwise, the exploration backtracks to test another solution. The maximal depth l can be reached is n which means a space complexity $O(n)$.

For example, as shown in Fig. 4, at depth 2, the subpermutation $(p(0) = 4, p(1) = 3)$ is acceptable by the consistency rule $d'_{p(i),p(j)} = d_{i,j}$ for each case (i, j) such that $0 \le i \le 1$ and $0 \le j \le 1$ (we use in this case distance matrix for graph representation), as follows:

$$\begin{pmatrix} 0 & 1 & | & | & | & | & | \\ 4 & 0 & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \end{pmatrix} \overrightarrow{R_p}\overrightarrow{C_p} \begin{pmatrix} | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & 0 & 4 & | & | \\ | & | & | & 1 & 0 & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \end{pmatrix}$$

However, when the algorithm goes in depth 3 by adding the new correspondence $p(2) = 0$ obtaining the subpermutation $(p(0) = 4, p(1) = 3, p(2) = 0)$, the consistency rule will be violated by $d_{2,0} \neq d'_{0,4}$ as follows:

$$\begin{pmatrix} 0 & 1 & | & | & | & | & | \\ 4 & 0 & | & | & | & | & | \\ \mathbf{3} & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \end{pmatrix} \overrightarrow{R_p}\overrightarrow{C_p} \begin{pmatrix} | & | & | & | & \mathbf{1} & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & 0 & 4 & | & | \\ | & | & | & 1 & 0 & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \end{pmatrix}$$

At this level when (11) is violated, the algorithm backtracks by excluding $p(2) = 0$ and goes another time in depth by adding the new correspondence $p(2) = 6$ which gives the subpermutation $(p(0) = 4, p(1) = 3, p(2) = 6)$ where (11) is respected as follows:

$$\begin{pmatrix} 0 & 1 & \mathbf{2} & | & | & | & | \\ 4 & 0 & \mathbf{6} & | & | & | & | \\ \mathbf{3} & \mathbf{1} & \mathbf{0} & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \end{pmatrix} \overrightarrow{R_p}\overrightarrow{C_p} \begin{pmatrix} | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ | & | & | & 0 & 4 & | & \mathbf{6} \\ | & | & | & 1 & 0 & | & \mathbf{2} \\ | & | & | & | & | & | & | \\ | & | & | & \mathbf{1} & \mathbf{3} & | & \mathbf{0} \end{pmatrix}$$

Note that cases which have been tested in previous depths (1 and 2) will not be tested again at the depth 3. Only cases $d_{i,j}$ of $i = 2$ or $j = 2$ must be seen whether they have a corresponding cases in D' under the considered subpermutation. As can be seen in Fig. 4 the in-depth exploration continues until the whole permutation satisfying (11) is detected.

In fact, in previous study, we supposed working with the ordered elements of the permutation $p(0),\ p(1),\ ...,\ p(n-1)$ to be associated to the *first*, *second*, *third*, 4th, ..., nth depth, respectively. This is not necessary, because it is possible to associate to the first depth any element $p(r)$ of p, and to the second depth another element $p(s)$ such that $r \neq s$.

In our algorithm, we propose this ordered elements of p : $p(k_1)$, $p(k_2)$, $..., p(k_n)$ such that $|S_{k_1}| \leq |S_{k_2}| \leq ... \leq |S_{k_n}|, k_i \in V$. Thus, to the *first*, *second*, *third*, 4th, ..., nth depth, are associated the elements $p(k_1)$, $p(k_2), ...,$ $p(k_n)$, respectively. In this way, it will be possible to figure out the most common subgraph even for graphs having different sizes.

Subgraph Isomorphism and Homomorphism

Subgraph isomorphism may be handled by CRGI2, after updating it, following the exact nature of the problem. For example, vertices partition must be tackled otherwise because the invariant notion will disappear in this case.

When two graphs of the same size are not isomorphic, the most common subgraph can be determined by CRGI2 which is defined by the subpermutation of the greater depth reached during the tree-search exploration.

For graphs $G = (V, E)$ and $G' = (V', E')$ having different sizes n and n', respectively ,where $n < n'$, obviously they cannot be isomorphic. To detect whether there exists a subgraph isomorphism from G to G', we search a subpermutation which associates for each vertex of V, a vertex from a subset S of V' preserving the property of the adjacency, i.e. $(i,j) \in E \Leftrightarrow (p(i), p(j)) \in E' \cap (S \times S)$. Therefore, it is sufficient to find for each case of the smaller matrix a corresponding case in the greater matrix by the subpermutation that satisfies (12). Thus, if the right subpermutation is found, the remainder cases of the greater matrix are not interesting in the subgraph isomorphism. As for the above-described process of exploration, and without changing anything in backtracking or tree pruning, it will be used to detect the subpermutation i.e. the correspondences from V to V'.

Actually, these are the main features that distinguish the subgraph isomorphism algorithm from the above described algorithm of graph isomorphism:

- The initial search space has a size of $\frac{n'!}{(n'-n)!}$. Suppose n is the size of the small graph G while the greater graph G' has n' vertices. In the first time, a vertex of G can be associated to any vertex among the n' vertices of G'. The second vertex of G, can be associated to $(n' - 1)$ vertices, and so on, until the last vertex of G that can be associated to $(n' - n + 1)$. Thus, the search space has a size of $n' \times (n' - 1) \times ... \times (n' - n + 1)$ i.e. $\frac{n'!}{(n'-n)!}$.
- No partition process does exists. In this case, the vertices of the two graphs have not the same properties like the invariant for graph isomorphism. However, it is possible to reduce the search space. A vertex of the small graph could be associated to vertices of the same or greater degree in the wider graph.
- Proposition (12) is used as consistency rule.
- A branch of the tree is a subpermutation $p = (p(0), ..., p(n - 1))$ which leads to a depth of n in the tree that represents the space complexity too. As for the exploration process still unchangeable.

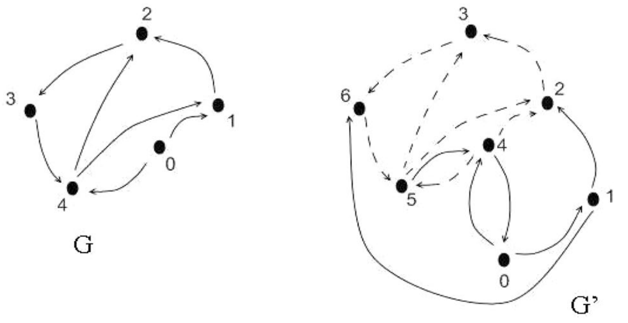

Fig. 5 Example of subgraph isomorphism

However, some algorithms deal with subgraph isomorphism taking into consideration the homomorphism. In this case, preserving the adjacency must not be in the two directions, but it is sufficient to be in only one direction, i.e. $(i,j) \in E \Rightarrow (p(i), p(j)) \in E' \cap (S \times S)$ as shown in Fig. 5 where edge (5, 4) of G' has not a corresponding edge in G i.e. $m'_{5,4} \neq m_{4,0}$. Therefore, the consistency rule is changed in this last case as used in [8] with adjacency matrices as follows:

$$\forall i,j \qquad (m_{i,j} = 1) \Rightarrow (m'_{p(i),p(j)} = 1) \tag{14}$$

which means if vertices i and j are linked by an edge in G, so their corresponding vertices $p(i)$ and $p(j)$ must be also linked. In fact, (13) represents the consistency rule for the homomorphism generally. Consequently, using (13) instead of (11) or (12) in CRGI2, homomorphism may be treated.

The below adjacency matrices M and M' represent G and G', respectively, given in Fig. 5

$$M = \begin{pmatrix} 0&1&0&0&1\\ 0&0&1&0&0\\ 0&0&0&1&0\\ 0&0&0&0&1\\ 0&1&1&0&0 \end{pmatrix}, M' = \begin{pmatrix} 0&1&0&0&1&0&0\\ 0&0&1&0&0&0&1\\ 0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1\\ 1&0&1&0&0&1&0\\ 0&0&1&1&\mathbf{1}&0&0\\ 0&0&0&0&0&1&0 \end{pmatrix}$$

Apparently, the tree exploration can find for the subgraph isomorphism shown in Fig. 5 by the following subpermutation:

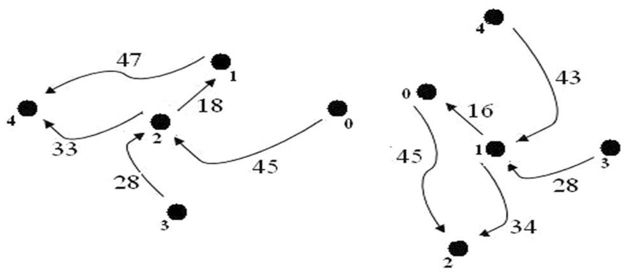

Fig. 6 Consistency rule for inexact graph matching

$$p = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 4 & 2 & 3 & 6 & 5 \end{pmatrix}$$

Applying the cases transformation on M by R_p and C_p, the obtained matrix $C_p(R_p(M))$ compared with M' shows that (13) is well satisfied. For each case of $C_p(R_p(M))$ containing '1', the same case in M' contains '1' too.

$$M = \begin{pmatrix} 0&1&0&0&1 \\ 0&0&1&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \\ 0&1&1&0&0 \end{pmatrix} \overrightarrow{R_p} \begin{pmatrix} | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ 0&0&1&0&0&|&| \\ 0&0&0&1&0&|&| \\ 0&1&0&0&1&|&| \\ 0&1&1&0&0&|&| \\ 0&0&0&0&1&|&| \end{pmatrix} \overrightarrow{C_p} \begin{pmatrix} | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ |&|&0&1&0&0&0 \\ |&|&0&0&0&0&1 \\ |&|&1&0&0&1&0 \\ |&|&1&1&\mathbf{0}&0&0 \\ |&|&0&0&0&1&0 \end{pmatrix}$$

An Inexact Matching for Attributed Graphs

Let G and G' be two graphs shown in Fig. 6 in which edges are attributed by real values. In this case, where matrix elements are real values, it is not demanded to respect the exact rule $x_{i,j} = x'_{p(i),p(j)}$, but it is sufficient to minimize the difference $|x_{i,j} - x'_{p(i),p(j)}|$ to find the optimal permutation of the matching. Several ways can be proposed to minimize $|x_{i,j} - x'_{p(i),p(j)}|$ following the context of use. For example, for image recognition, it may be introduce some error-tolerant method to tackle the image deformation that appears in matrix representing an image.

The following matrices M and M' are representing graphs G and G' given in Fig. 6.

$$M = \begin{pmatrix} 0&0&45&0&0 \\ 0&0&0&0&47 \\ 0&18&0&0&33 \\ 0&0&28&0&0 \\ 0&0&0&0&0 \end{pmatrix}, M' = \begin{pmatrix} 0&0&45&0&0 \\ 16&0&34&0&0 \\ 0&0&0&0&0 \\ 0&28&0&0&0 \\ 0&43&0&0&0 \end{pmatrix}.$$

In our case, instead using (11) it will be possible to use the consistency rule:

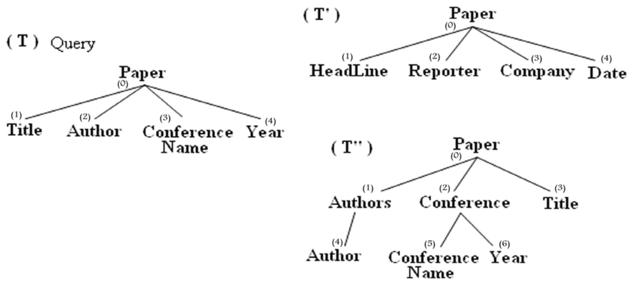

Fig. 7 Trees for an XML query (T) and two XML documents (T') (T")

$$\forall i,j \qquad |x_{i,j} - x'_{p(i),p(j)}| \le \epsilon, \tag{15}$$

where ϵ is error-tolerance threshold of the matching. Obviously, whenever ϵ is close to zero, the graphs are more matched. Notice that to determine the optimal permutation of the matching, the process of exploration in the tree-search still unchangeable. Making $\epsilon = 2(4\%)$, it can be seen through Fig. 6 that the permutation

$$p = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 4 & 0 & 1 & 3 & 2 \end{pmatrix},$$

gives the following cases transformation:

$$M = \begin{pmatrix} 0&0&45&0&0 \\ 0&0&0&0&47 \\ 0&18&0&0&33 \\ 0&0&28&0&0 \\ 0&0&0&0&0 \end{pmatrix} \overrightarrow{R_p} \begin{pmatrix} 0&0&0&0&47 \\ 0&18&0&0&33 \\ 0&0&0&0&0 \\ 0&0&28&0&0 \\ 0&0&45&0&0 \end{pmatrix} \overrightarrow{C_p} \begin{pmatrix} 0&0&47&0&0 \\ 18&0&33&0&0 \\ 0&0&0&0&0 \\ 0&28&0&0&0 \\ 0&45&0&0&0 \end{pmatrix}$$

which satisfies $|m_{i,j} - m'_{p(i),p(j)}| \le 2$, for all cases.

Tree Matching for XML Retrieval

One of the most studied problems in the information retrieval field is the querying XML documents [28]. In this case, XML documents are considered having a tree structure[4] and the query is a subtree like is shown in Fig. 7. As it is known, trees are a special kind of graphs such that there is no cycles. The aim is to find whether a subtree (query) is included in one of the collection of trees (XML documents). It is clear that there is some form of matching between T and T'', whereas T' is not relevant to the others. In this case, some constraints are posed (i) for vertices, each vertex of the query must appear in the XML Document (ii) as for edges, each related vertices in the query

[4] It is worth to make difference between tree as graph and tree as a search method.

(arc: parent-child) can be matched by a pair of vertices linked by a path (ancestor-descendant), for example edge (*Paper*, *Year*) of the query could be associated with the path (*Paper*, ..., *Year*) in T''.

Obviously, the adjacency matrix cannot be used here, however, the distance matrix is able to deal with such kind of matching. When a query contains two vertices: parent–child i.e. ($d_{i,j} = 1$), it is sufficient to find corresponding vertices (ancestor-descendant) where ($d'_{p(i),p(j)} \geq 1$). Therefore, we say:

$$\forall d_{i,j} \in D, \qquad (d'_{p(i),p(j)} \geq d_{i,j}) \tag{16}$$

If we use the following distance matrices as representations of (T) and (T''):

$$D = \begin{pmatrix} Paper & 1 & 1 & 1 & 1 \\ 0 & Title & 0 & 0 & 0 \\ 0 & 0 & Author & 0 & 0 \\ 0 & 0 & 0 & C.Name & 0 \\ 0 & 0 & 0 & 0 & Year \end{pmatrix}$$

D’=

$$\begin{pmatrix} Paper & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & Authors & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & Conf & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & Title & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & Author & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & C.Name & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & Year \end{pmatrix}$$

From diagonal cases of the matrices, the search space is reduced which leads to the subpermutation:

$$p = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 0 & 3 & 4 & 5 & 6 \end{pmatrix}$$

that can give a possible matching if the constraints are not violated. The matrix $C_p(R_p(D))$ below shows that (15) is well respected.

$$C_p(R_p(D)) = \begin{pmatrix} Paper & | & | & 1 & 1 & 1 & 1 \\ | & | & | & | & | & | & | \\ | & | & | & | & | & | & | \\ 0 & | & | & Title & 0 & 0 & 0 \\ 0 & | & | & 0 & Author & 0 & 0 \\ 0 & | & | & 0 & 0 & C.Name & 0 \\ 0 & | & | & 0 & 0 & 0 & Year \end{pmatrix}$$

As it was seen above, it will be possible to adapt CRGI2 for querying XML documents using its backtracking exploration to find the subpermutation of the matching integrating, of course, the consistency rule expressed in (15).

Experimental Results

Experiments described in this section are conducted on two data sets: a synthetic data set [29] and a real world data set [30]. First, an evaluation of the performance is realized through a benchmarking process to show the efficiency of CRGI2 and other known algorithms for the graph isomorphism problem. Second, the flexibility of CRGI2 is illustrated for subgraph isomorphism and its applicability to information retrieval tasks of relational data.

A Performance Comparison for Graph Isomorphism

In this section, CRGI2 is tested for graph isomorphism and its performance is compared with other known algorithms of exact matching, namely VF, VF2, SD and Ull (Ullmann). Details on these kinds of algorithms can be found in [7–9]. Algorithm CRGI2 has been executed in two forms: in the first CRGI2(Adj), the graphs are represented by adjacency matrix using degree as invariant vertices to determine partitions of vertices. The second form of execution CRGI2(Dis) uses distance matrix as graph representation and vector of distances as invariant (for each node, the number of vertices reachable at each distance is calculated).

For the remainder algorithms VF, VF2, SD, and Ull, we have used the library VFLib2.0 [7, 32] that contains the implementation of these algorithms. In this sense, CRGI2 was implemented in C++ like the other algorithms. All the programs have been compiled with the same compiler.

The evaluation is carried out using a collection of isomorphic graphs that contains 9000 couples. A part of this collection is containing in the database realized by [29] while the rest of the collection has been generated by us which represents trees and graphs having sizes greater than 1000 vertices. This collection is distributed on the following kinds of graphs: *randomly connected graphs* (2200 couples), regular and irregular *2D meshes* (2000 couples), irregular *bounded valence graphs* (2000 couples) and *random trees* (2800 couples). Each category includes couples of isomorphic graphs of different sizes, ordered from dozens of vertices to about a few thousands of vertices. Each size and kind of graphs has 100 different couples except graphs of thousands of vertices, where only 10 couples that exist.

The performance is evaluated through a benchmarking process in which each algorithm tries to detect the isomorphism permutation (with the necessary time) of each pair among the 9000 existing couples. For each isomorphic couples (of the same kind and size), the average time of exploring the tree-search to detect the permutation is calculated.

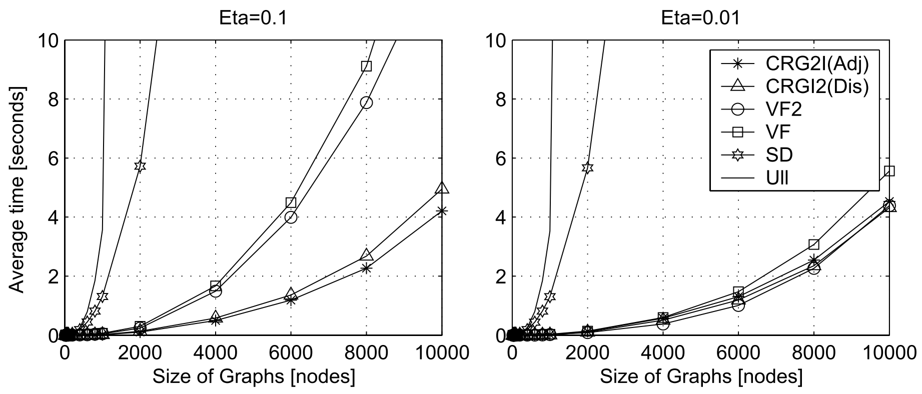

Fig. 8 Performance on Randomly Connected Graphs with $\eta = 0.01$ and $\eta = 0.1$

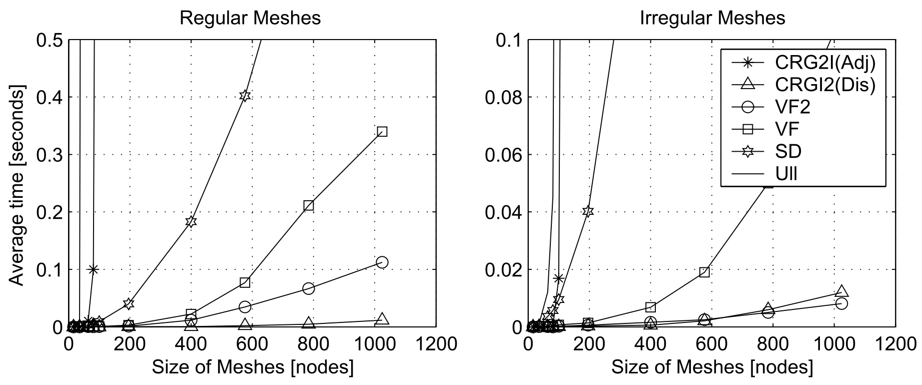

Fig. 9 Performance on 2D Meshes (Regular and Irregular meshes)

Randomly Connected Graphs

Figure 8 shows, for each size of graphs, the average time of exploring the tree-search to detect the isomorphism permutations for the selected algorithms, using randomly connected graphs for values of density η equal to 0.01 and 0.1 described in [29].

It can be seen that the two versions of CRGI2 and the two VF perform always better than SD and Ull. After a size of 2000, the SD algorithm requires an exponential time to detect any isomorphism while the Ull algorithm cannot deal with graphs of more than 1000 vertices. For CRGI2, either using adjacency or distance matrices, the two versions behave nearly in the same manner. CRGI2 is independent from the density of the graph where about less than 6 seconds are needed for discovering the isomorphism for graphs of 10,000 vertices. Conversely, the two VF behaves in different manners following the values of density.

2D Meshes

In this case, the behavior of the two version of CRGI2 is not the same. CRGI2(Adj) gives the worst performance with Ullmann's algorithm as shown in Fig. 9. Either meshes are regular or irregular having sizes greater than 100 vertices, CRGI2(Adj) requires an exponential time to figure out the isomorphism. Paradoxically, CRGI2(Dis) gives better results than the other algorithms. This wide discard in behavior shows clearly that choosing the parameters of CRGI2 has an important factor to deal efficiently with graph isomorphism problem.

Bounded Valence Graphs

The performance of the algorithms on bounded valence graphs are shown in Fig. 10. In this case, the considered values of the valence are 3 and 9 described in [29].

For valence equal to 3, graph isomorphism is hard to be treated by most of algorithms except VF2 that is able to find isomorphism for all graphs in less than 1 s. In this case, VF and VF2 perform better than both CRGI2(Dis) and CRGI2(Adj). The first is not able to find a solution when the size grows up to 600 vertices while the second cannot deal with graphs of more than 200 vertices. However for the graphs of valence 9, the two execution of CRGI2 becomes more efficient.

Random Trees

We attempt to test in this section trees as graphs to show the flexibility of these algorithms against the various kinds of graphs.

The random trees are generated as follows: each vertex has a random number of children referred as *nbr_child*

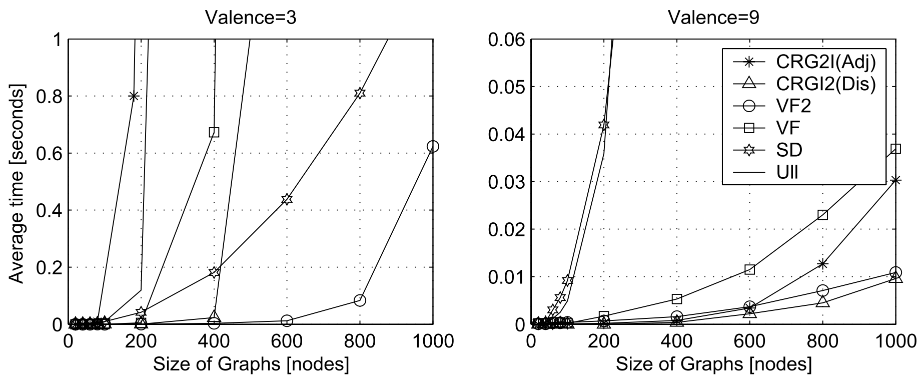

Fig. 10 Performance on Bounded Valence Graphs

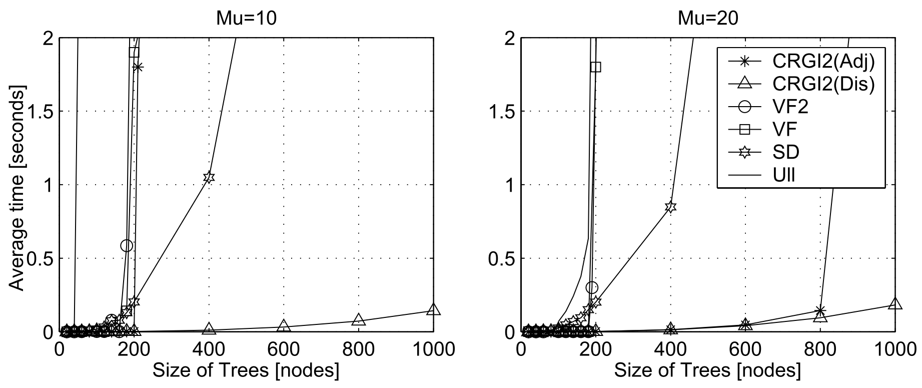

Fig. 11 Performance on Random Trees

Table 1 Average time [in s] to detect isomorphism of Random Trees ($\mu = 10$)

Tree Size	CRGI2 (Adj)	CRGI2 (Dis)	VF2	VF	SD	Ull
20	0	0	0	0	0.001	0
40	0.0001	0.0001	0	0	0.005	0.004
60	0	0	0.0005	0.0005	0.004	?
80	0.0002	0.0001	?	?	0.009	?
100	0.0002	0.0002	0.0015	0.002	0.019	?
120	0.0001	0.0007	0.0015	0.0005	0.032	?
140	0.0009	0.0006	0.0785	?	0.075	?
160	0.001	0.001	0.001	?	0.078	?
180	0.0014	0.0018	0.585	0.142	0.129	?
200	0.0018	0.0018	?	?	0.202	?
400	22.232	0.0108	?	?	1.049	?
600	?	0.0315	?	?	3.691	?
800	?	0.0718	?	?	7.1265	?
1000	?	0.1428	?	?	12.699	?

between 0 and μ. In our case, μ is fixed by two values 10 and 20.

Comparison process in Fig. 11 shows that CRGI2(Dis) performs better than any other algorithm. Tables 1 and 2 and show that only CRGI2(Dis) and SD are able to find solutions for all random trees. However, algorithms Ull, VF and VF2 cannot detect isomorphism during one hour (represented by ?) for some trees that have sizes not exceeding 100 vertices.

Enron Database

The Enron corpus contains a large set of e-mail messages belonging to users of the Enron corporation [30, 31]. The underlying data is converted into one directed graph by representing the senders and the receivers as vertices, attributed with their corresponding e-mail address. Solely e-mail addresses having @enron.com as suffix are taking

Table 2 Average time [in s] to detect isomorphism of Random Trees ($\mu = 20$)

Tree Size	CRGI2 (Adj)	CRGI2 (Dis)	VF2	VF	SD	Ull
20	0	0	0	0.0001	0.0003	0.0003
40	0.0002	0	0	0	0.0013	0.002
60	0	0.0001	0	0	0.0061	0.0091
80	0.0001	0.0001	?	?	0.0107	?
100	0.0004	0.0003	0.0001	0	0.0249	0.0685
120	0.0004	0.0004	0.0002	0.0006	0.0462	0.1388
140	0.0009	0.0007	0.0001	0.0003	0.0688	0.2419
160	0.0013	0.0013	0.0002	0.0005	0.0948	0.3758
180	0.002	0.0011	0.0003	0.0004	0.1503	0.6344
200	0.002	0.0012	?	?	0.2008	?
400	0.0142	0.0131	?	?	0.8487	?
600	0.0464	0.041	?	?	4.657	?
800	0.144	0.0974	?	?	5.91	?
1000	?	0.182	?	?	16.998	?

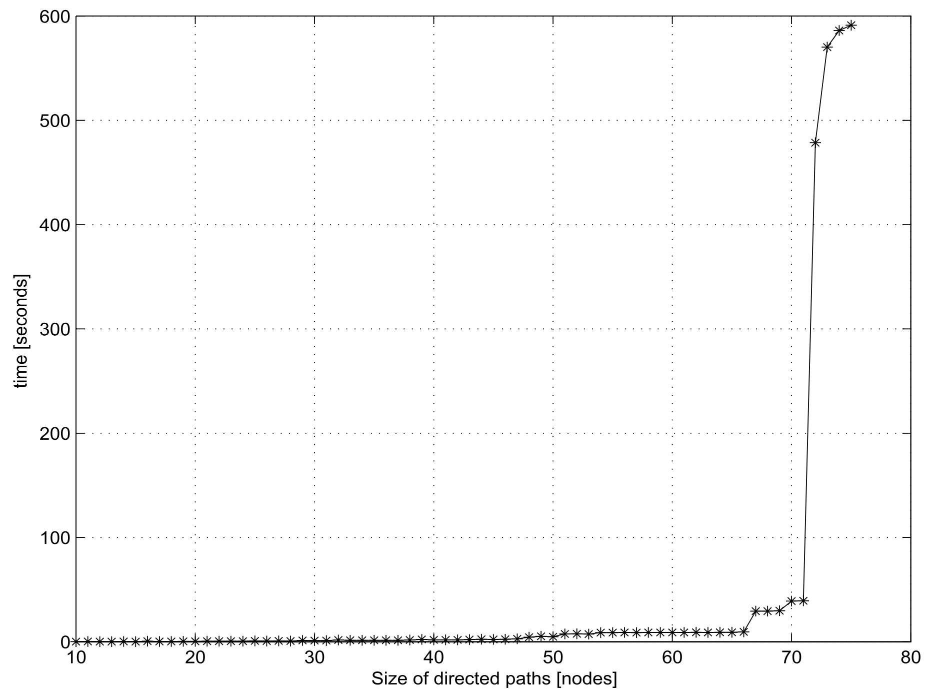

Fig. 12 Time for discovering directed paths in the Enron graph

into account. If there is at least a message from a vertex to another, so an edge is used as a link. Only messages from one to one are considered i.e. diffusion from one to many is ignored. The complete data set gives a directed graph of 7592 vertices and 31,789 edges.

Therefore, it will be possible to test some queries on the Enron graph. For example, the maximal clique in the Enron graph is a complete subgraph of 7 vertices attributed with {mary.cook, elizabeth.sager, peter.keohane, louise.kitchen, marie.heard, tana.jones}@enron.com. This clique is detected in 0.12 s using CRGI2 that checks first the subgraph isomorphism from a complete graph of n vertices to the Enron graph without attributes. The maximum value of n can be reached is 7, while the given subpermutation of the subgraph isomorphism leads easily to the e-mail addresses of each node.

In the same manner, Fig. 12 shows how GRGI2 can detect directed paths not exceeding 80 vertices in the Enron graph. For paths smaller than 60 vertices, a few seconds less than 10 are needed to find a path in the Enron graph. Unfortunately, when the paths grow up to 70 vertices and greater, the necessary time of exploration increases dramatically. For example, a path of 75 vertices is discovered in 591.23 seconds, while a path of 76 vertices cannot be discovered. Thus,

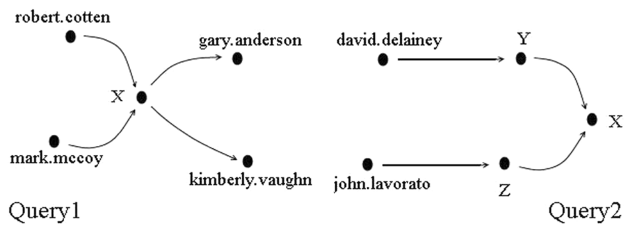

Fig. 13 Query graphs to the Enron database graph

we do not know if there is a path of 76 vertices, because the algorithm is unable to give a response.

However, other queries on the Enron graph are so important for information retrieval, for example in Fig. 13 two queries are illustrated. The first query corresponds to the question: which e-mail address X has received messages from {robert.cotten and mark.mccoy}@enron.com and then has sent messages to both {gary.anderson and kimberly.vaughn}@enron.com? For the second query, the corresponding question is: which e-mail address X has received messages from Y and Z, where Y and Z have received also messages from {david.delainey and john.lavorato}@enron.com? (see Fig. 13)

By applying CRGI2 as a subgraph isomorphism algorithm knowing that some vertices are attributed (an attributed vertex in the query must be associated to a vertex that has the same attribute), CRGI2 has discover, in 2.21 s, only one subgraph in the Enron graph which is isomorphic to the first query where X = clem.cernosek@enron.com.

For the second query, CRGI2 has found seven possible matchings during 70.67 s. The values of X, Y and Z, without @enron.com, are as follows:

1. X = marchris.robinson, Y = steven.kean, Z = kay.mann
2. X = steve.montovano, Y = steven.kean, Z = kay.mann
3. X = janet.dietrich, Y = steven.kean, Z = kay.mann
4. X = brian.redmond, Y = steven.kean, Z = kay.mann
5. X = janette.elbertson, Y = richard.b.sanders, Z = kay.mann
6. X = rob.walls, Y = steven.kean, Z = kay.mann
7. X = rob.walls, Y = richard.b.sanders, Z = kay.mann

Conclusion

The proposed algorithm deals with graphs without making any particular assumptions on the structure of the used graphs, in contrast to the other algorithms that require at least connected graphs. Note also that, theoretically, there is no condition on the upper bound for the number of vertices (or the number of edges) in order to solve the isomorphism equation $M' = P^tMP$. Unfortunately, since there is no mathematical method to solve $M' = P^tMP$ except using exhaustive search methods, it will be a difficult task (and even not possible) to find an isomorphism for graphs having high number of vertices. The upper bound will be certainly depends on : (1) the power of calculus of the machine, (2) the used algorithm itself, and (3) the type of graph. Our experiments show the ability of solving isomorphism permutation for random graphs of 10,000 vertices. However, for other types of graphs, no result can be found for considerable values of $n < 1000$. Globally, no algorithm succeed to detect the graph isomorphism in polynomial time. There is at least one case where every algorithm fails due to the explosion of calculations.

In general, although the proposed algorithm is based on a simple consistency rule, it shows its efficiency and illustrates to be suitable for various matching problems. The simplicity of the consistency rule itself leads to accelerating calculations and to be applied for a collection of matching problems.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Bunke H. On a relation between graph edit distance and maximum common subgraph. Pattern Recognit Lett. 1997;18(8):689–94.
2. Fernández ML, Valiente G. A graph distance metric combining maximum common subgraph and minimum common supergraph. Pattern Recognit Lett. 2001;22(6–7):753–8.
3. Qiu H, Hancock ER. Graph matching and clustering using spectral partitions. Pattern Recognit. 2006;39(1):22–34.
4. Umeyama S. An Eigen decomposition approach to weighted graph matching problems. IEEE Trans Pattern Anal Mach Intell. 1988;10(5):695–703.
5. Lopez-Presa JL, Fernandez Anta A. Fast algorithm for graph isomorphism testing. In: Proceedings of the 8th International Symposium on experimental algorithms, Springer-Verlag, Dortmund, Germany, June 4–6, 2009, p. 221–32.
6. Messmer B, Bunke H. A decision tree approach to graph and subgraph isomorphism detection. Pattern Recognit. 1999;32(12):1979–98.
7. Cordella LP, Foggia P, Sansone C, Vento M. A (sub)graph isomorphism algorithm for matching large graphs. IEEE Trans Pattern Anal Mach Intell. 2004;26(10):1367–72.
8. Ullmann JR. An algorithm for subgraph isomorphism. J ACM. 1976;23(1):31–42.
9. Schmidt DC, Druffel LE. A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. J ACM. 1976;23(3):433–45.
10. Benreguia B, Kheddouci H. A consistency rule for graph isomorphism problem. In: Proceedings of the 27th Annual ACM Symposium on applied computing, ACM, 2012; p. 906–11.
11. McKay B. Practical graph isomorphism. Congr Numer. 1981;30:45–87.

12. Chan SC, Cheney J. Flexible Graph Matching and Graph Edit Distance Using Answer Set Programming. In: International Symposium on Practical Aspects of Declarative Languages PADL 2020, Lecture Notes in Computer Science, Springer, Cham, 2020;12007:20–36.
13. Bougleux S, Gauzere B, Brun L. A Hungarian algorithm for error-correcting graph matching. In: International Workshop on graph-based representations in pattern recognition. GbRPR 2017. Lecture Notes in Computer Science, Springer, Cham, 2017;101310:118–127.
14. Boria N, Blumenthal DB, Bougleux S, Brun L. Improved local search for graph edit distance. Pattern Recognit Lett. 2020;129:19–25.
15. Kataoka T, Shiotsuki E, Inokuchi A. Graph Classification with Mapping Distance Graph Kernels. In: International Conference on Pattern recognition applications and methods. ICPRAM 2017. Lecture Notes in Computer Science, Springer, Cham, 2017;10857:21–44.
16. Kausar A, Enrico A, Diana S, Uttara T, Duy D, Mintao L, Meenusree R, Jaroslaw H, Beau A, Joaquín G. GEFF: graph embedding for functional fingerprinting, NeuroImage, 2020;10857:117181.
17. Mukhopadhyay A, Kumar ACS, Bhandarkar SM. Joint geometric graph embedding for partial shape matching in images. In: IEEE Winter Conference on applications of computer vision (WACV), Lake Placid, NY, USA, March 7–10, 2016, p. 1–9.
18. Demirci MF, Kacka S. Object recognition by distortion-free graph embedding and random forest. In: IEEE Tenth International Conference on semantic computing (ICSC), Laguna Hills, California, USA, 4–6 February 2016, p. 17–23.
19. Zavlanos MM, Pappas GJ. A dynamical systems approach to weighted graph matching. Automatica. 2008;44:2817–24.
20. Kang U, Hebert M, Park CS. Fast and scalable approximate spectral graph matching for correspondence problems. Inf Sci. 2013;220:306–18.
21. Zanfir A, Sminchisescu C. Deep Learning of graph matching. In: The IEEE Conference on computer vision and pattern recognition (CVPR), Salt Lake City, UT, USA, June 18–22, 2018, p. 2684–693.
22. Guo M, Chou E, Huang D, Song S, Yeung S, Fei-Fei L. Neural graph matching networks for fewshot 3D action recognition. In: The European Conference on computer vision (ECCV), Munich, Germany, September 8–14, 2018, p. 653–69.
23. Riesen K, Jiang X, Bunke H. Exact and inexact graph matching: methodology and applications, In: Aggarwal C., Wang H. (eds) Managing and Mining Graph Data. Advances in Database Systems, Springer, Boston, MA, 2010;40:217–47.
24. Yan J, Yin X, Lin W, Deng C, Zha H, Yang X. A short survey of recent advances in graph matching. In: Proceedings of the 2016 ACM on International Conference on multimedia retrieval, ICMR 2016, New York, New York, USA, June 6–9, 2016, p. 167–74.
25. Livi L, Rizzi A. The graph matching problem. Pattern Anal Appl. 2013;16:253–83.
26. Conte D, Foggia P, Sansone C, Vento M. Thirty years of graph matching in pattern recognition. Int J Pattern Recognit Artif Intell. 2004;18(03):265–98.
27. Ferrer B, Bunke H. Graph edit distance-theory, algorithms, and applications. Image Processing and Analysis with Graphs: Theory and Practice (1st ed.). CRC Press - Taylor & Francis Group, Boca Raton, FL, USA, 2012, p. 384–422.
28. Jeong B, Lee D, Cho H, Lee J. A novel method for measuring semantic similarity for xml schema matching. Expert Syst Appl. 2008;34(3):1651–8.
29. Santo MD, Foggia P, Sansone C, Vento M. A large database of graphs and its use for benchmarking graph isomorphism algorithms. Pattern Recognit Lett. 2003;24(8):1067–79.
30. Klimt B, Yang Y. Introducing the enron corpus,. In: First Conference on email and anti-spam (CEAS), Mountain View, California, USA, July 30-31, 2004.
31. Brugger A, Bunke H, Dickinson P, Riesen K. Generalized graph matching for data mining and information retrieval. In: Proceedings of the 8th Industrial Conference on advances in data mining: medical applications, E-commerce, marketing, and theoretical aspects. Springer-Verlag; 2008, p. 298–312.
32. Foggia P, Sansone C, Vento M. A performance comparison of five algorithms for graph isomorphism. In: Proceedings of the 3rd IAPR TC- 15 Workshop on Graph-based representations in pattern recognition, Ischia, Italy, May 23–25, 2001, p. 188–199.