Multivalent Graph Matching for Symbol Recognition

D. K. Ho(✉), J. Y. Ramel, and N. Monmarché

Laboratoire d'Informatique Fondamentale et Appliquée de Tours, Université de Tours, Tours, France
{kieu.ho,jean-yves.ramel,nicolas.monmarche}@univ-tours.fr

Abstract. This paper deals with a symbol recognition problem where symbols are deformed in different ways. To address this difficulty, the multivalent graph matching problem is considered to allow more matching possibilities between the symbols. More precisely, a solution to the multivalent graph matching problem is formulated as an extended graph edit distance minimum with additional splitting and merging operations. This minimization problem is tackled with a variant of ant colony optimization, the max-min ant system. Besides, a neighborhood search strategy added to the solution building process of the max-min ant system aims to accelerate the computational time. The efficiency of the proposed approach is illustrated in a symbol dataset in several aspects, covering both quality and quantity analyses. The result shows the interest in using multivalent graph matching to deal with noisy symbols and their meaningful interpretability in the context of sub-part correspondence against other bijective graph matching-based approaches.

Keywords: Multivalent graph matching · Extended graph edit distance · Max-min ant system · Neighbor search · Symbol recognition · Classification

1 Introduction

Several applications in real life demand the determination of an explainable similarity measure between the objects rather than a numerical value. This is the case in biometric identification, symbol recognition, medical diagnostics or handwritten document analysis [5,6,11,13,20,23]. Among the existing approaches, the graph-based approach is promising because it provides the sub-part correspondence as well as the similarity measure. In particular, a first step called graph representation is done to transform the compared objects into the corresponding graphs based on the features and topologies. Then, the problem of defining the similarity of objects turns into the problem of matching graphs regarding the features of vertices and edges of the graphs. Within some domains such as 2D and 3D image analysis, biological and biomedical applications, we can find applications of graph-based approach [8,18,20,22]. Those strengthen the power of graph-based techniques in digital society nowadays.

A Graph Matching problem (GM) belongs to either exact GM or inexact GM. For exact GM, the matching between vertices and edges must respect both the attributes and the structure of the graph [7]. Consequently, it is well adapted to symbolic attributes and in a non-noisy context. Inexact GM is more flexible since it can violate the previous constraints about the attributes or the structure. In the family of inexact GM, Graph Edit Distance (GED) is a way to produce inexact GM. Solving the GED problem leads to minimize the distance or dissimilarity measure between two graphs through a series of edit operations. Regularly, these edit operations are substitution, deletion, insertion of vertices or edges, with a specific cost associated to each operation. Two main categories of methods, exact and heuristic, are proposed to tackle the GED. Unfortunately, the exact methods like A^* [9] have expensive computation time. Meanwhile, the heuristic methods like A^*-Beam Search [12] have lower computational time with acceptable solution quality. Furthermore, the exact methods seem to be infeasible for solving the GM problem with large graph size from more than 100 nodes in an acceptable time. Therefore, more and more heuristic approaches are developed for GED problem.

According to [19], the multivalent GM problem is more general than the GED because it permits one vertex in one graph matching with more than one correspondence in the other graph. In some cases, one feature in one object can correspond to multiple ones in the other object [1,4]. Furthermore, in pattern recognition problems, distortions of the graph can occur, and error-tolerant graph matching techniques should be used to allow node (edge) association even if they are not absolute similar [7]. All these matching situations are special cases of multivalent matching. Therefore, multivalent GM can be seen as the most general GM problem, and we will contribute to bring the solution to solve it in this work.

While reviewing the literature, we found some interesting works dealing with the multivalent GM problem. In [1], the authors apply the GED-based GM technique for recognizing diatoms. However, only merging and splitting nodes are completed in addition to the classic operations to accommodate the problem. In [3], the authors also suggest a GED-based approach with node merging to evaluate the similarity between images. Likewise in the previous works, the edge relations related to the fusion of nodes are still unclear. In contrast, in works [2,3], the authors specify the edge merging when it is necessary. However, the experimental results are not compared to other methods. In [4], a non-bijective GM approach is utilized for seeking the correspondence between an original image and its over-segmented ones. Due to the problem context, it is a bit confused between node/edge substitution and node/edge merging. In [16,17], the authors propose a general similarity measure for the multivalent GM and employ Ant Colony Optimization (ACO), reactive tabu search to tackle the problem. However, the measure, which is intended for graphs with symbolic attributes, restricts the scope of applications. In [10], the authors address the multivalent GM problem as an extension of GED called Extended Graph Edit Distance (ExGED). A formal formulation of ExGED based on the concept of a cost matrix is given, and they provide a way to calculate the costs of merging/splitting edges in extended cases. Nonetheless, the computational time is a drawback of the approach.

From the mentioned works, we see that the GED-based approach is popular for a multivalent GM problem. Following the direction of the work [10], we formalize the multivalent GM problem as an ExGED one and specifying the costs for edge merging and splitting induced by the edit operations done on nodes. These costs are integrated into a cost matrix that considers both semantic and topological features. We also apply the ACO-based methods, which is Max-Min Ant System (MMAS), to resolve the problem. But, we enhance the performance of MMAS by reducing its computational time. Particularly, a neighbor search strategy based on the previously selected nodes is applied. Hence, the number of candidates for the next steps will be shrunked and accelerating the solution construction process of MMAS. The effectiveness of the proposed approach is presented through numerical experiments for the symbol recognition problem. The results show a significant improvement in MMAS's execution time while reserving a good solution quality.

The paper is organized as follows. Firstly, a problem formulation from GED to ExGED is presented in Sect. 2. Secondly, the cost matrix construction for ExGED is detailed in Sect. 3. Thirdly, the MMAS algorithm and the strategy to reduce its computational time are specified in Sect. 4. Next, the efficiency of the proposed approach is shown in Sect. 5. Finally, the conclusion sums up the principal points and some suggestions in the future.

2 From GED to ExGED Problem Formulation

2.1 Graph Edit Distance

Definition 1. *An attributed graph contains 4-tuple $G = (V, E, \mu, \xi)$, where*

V, E are sets of vertices and edges, respectively,
l_V, l_E are sets of vertex and edge labels, respectively
$\mu : V \mapsto l_V$: function that assigns labels to vertices
$\xi : E \mapsto l_E$: function that assigns labels to edges.

Definition 2. *An edit path is a sequence of edit operations (ed_i) to transform one graph to another graph, denoted $\lambda(G_1, G_2) = \{ed_i\}$. A valid edit path should follow these conditions: 1) deleting a vertex implies deleting its related edges; 2) inserting an edge is only permitted if the two vertices already exist; 3) inserting an edge must not create more than one edge between two vertices (selfloops) [7].*

Definition 3. *Given two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, the graph edit distance (GED) is a dissimilarity measure between G_1 and G_2 and is defined by:*

$$d_{min}(G_1, G_2) = \min_{\lambda \in \Theta(G_1, G_2)} \sum_{ed_i \in \lambda} c(ed_i), \tag{1}$$

where $\Theta(G_1, G_2)$ is the set containing all valid edit paths λ between G_1 and G_2, $c(ed_i)$ is the cost of each edit operation ed_i [7].

Classical edit operations are given in Table 1. The cost of each operation is defined according to either the node or edge labels.

2.2 Extended Graph Edit Distance Problem Formulation

Extended graph edit distance (ExGED) is also a dissimilarity measure derived from the costs of the edit operations. In addition to traditional edit operations in GED, the splitting and merging operations are supplemented to consider multivalent matching as follows.

Definition 4. *Given two attributed graphs* $G_1 = (V_1, E_1, \mu_1, \xi_1)$ *and* $G_2 = (V_2, E_2, \mu_2, \xi_2)$*, we define:*

- *merging the set* $S_{mer} = \{u_i \in V_1, i \geq 2\}$ *to* $v \in V_2$ *is noted* $S_{mer} \rightarrow v$
- *splitting* $u \in V_1$ *into the set* $S_{spl} = \{v_j \in V_2, j \geq 2\}$ *is noted* $u \rightarrow S_{spl}$

These two operators are also mentioned in Table 1. Usually, doing node merging and splitting can lead to edge splitting and merging, this will be discussed more precisely when the cost of each operation will be detailed.

Table 1. Availability of edit operations for GED and ExGED (with $u \in V_1, v \in V_2, e_1 \in E_1, e_2 \in E_2, \varepsilon$ the *virtual* vertex or edge). S_{mer} and S_{spl} are subsets of V_1 and V_2 defined in Definition 4.

Operation	Notation	Cost function notation	GED	ExGED
Vertex substitution	$u \rightarrow v$	$c(u \rightarrow v)$	✓	✓
Vertex deletion	$u \rightarrow \varepsilon$	$c(u \rightarrow \varepsilon)$	✓	✓
Vertex insertion	$\varepsilon \rightarrow v$	$c(\varepsilon \rightarrow v)$	✓	✓
Edge substitution	$e_1 \rightarrow e_2$	$c(e_1 \rightarrow e_2)$	✓	✓
Edge deletion	$e_1 \rightarrow \varepsilon$	$c(e_1 \rightarrow \varepsilon)$	✓	✓
Edge insertion	$\varepsilon \rightarrow e_2$	$c(\varepsilon \rightarrow e_2)$	✓	✓
Vertex merging	$S_{mer} \rightarrow v$	$c(S_{mer} \rightarrow v)$		✓
Vertex splitting	$u \rightarrow S_{spl}$	$c(u \rightarrow S_{spl})$		✓

3 Cost Matrices for ExGED

3.1 Definition of the Cost Matrix for Node Operations

Inspired by the idea in [15], a cost matrix for ExGED is also built with five types of edit operations. These operations are organized in separate blocks along with their corresponding costs. Formally, given two attributed graphs G_1, G_2 as above, we denote that $n = |V_1|, m = |V_2|$. $P^k_{mer} = \{S^i_{mer}\}$ is a set of all possibilities for merging of nodes in G_1 and $h = |P^k_{mer}|$. $P^k_{spl} = \{S^j_{spl}\}$ is a set of all possibilities for splitting of nodes in G_2 and $l = |P^k_{spl}|$. k is a parameter that describes the maximum size of sets S^i_{mer} and S^j_{spl}, $k \geq 2$. The cost matrix is demonstrated as below.

$$\mathbf{C} = \begin{array}{ccc|c|ccc|c}
1 & \dots & m & \varepsilon & 1 & \dots & l & \\ \hline
c_{1,1} & \dots & c_{1,m} & c_{1,\varepsilon} & c_{1,S^1_{spl}} & \dots & c_{1,S^l_{spl}} & 1 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
c_{n,1} & \dots & c_{n,m} & c_{n,\varepsilon} & c_{n,S^1_{spl}} & \dots & c_{n,S^l_{spl}} & n \\ \hline
c_{\varepsilon,1} & \dots & c_{\varepsilon,m} & 0 & \infty & \dots & \infty & \varepsilon \\ \hline
c_{S^1_{mer},1} & \dots & c_{S^1_{mer},m} & \infty & \infty & \dots & \infty & 1 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
c_{S^h_{mer},1} & \dots & c_{S^h_{mer},m} & \infty & \infty & \dots & \infty & h
\end{array}$$

where $c_{i,j}$ denotes the cost of a node substitution (with $(i,j) \in \{1 \dots n\} \times \{1 \dots m\}$), $c_{i,\varepsilon}$ denotes the cost of a node deletion, $c_{\varepsilon,j}$ denotes the cost of a node insertion, $c_{i,S_{spl}}$ denotes the cost of a node splitting and $c_{S_{mer},j}$ denotes the cost of a node merging.

The cost matrix $\mathbf{C}$ is not a square matrix as in GED case because of dimension reductions on deletion block $((n \times n) \rightarrow (n \times 1))$ and insertion one $((m \times m) \rightarrow (1 \times m))$. This dimensional reduction aims to save more memory but still preserving the property of a GED cost matrix. Moreover, by introducing splitting and merging operations, the size of matrix $\mathbf{C}$ increases with h rows and l columns. h and l are strongly influenced by k. The bigger k is the higher values of h, l get. Consequently, the size of $\mathbf{C}$ will grow up significantly. Thus, choosing the number of k would be very important, especially in a big graph. Regularly, the parameter k is problem-dependent and based on expert knowledge.

3.2 Definition of the Costs for Edge Operations in Extended Case

To keep the computational cost reasonable, we propose to integrate the estimated costs of edge operations involved in each node operation inside the previous matrix $\mathbf{C}$. Consequently, we need to estimate a cost for the edge operations in the extended case. The detail is given below.

- For node substitution $u_i \rightarrow v_j$, two sets of incident edges of u_i and v_j are computed, called E_{u_i} and E_{v_j}, respectively. Then, a square edge cost matrix $\mathbf{Ce}$ is built from E_{u_i} and E_{v_j} based on the cost functions of edge operations in Table 1. The size of $\mathbf{Ce}$ is $(|E_{u_i}| + |E_{v_j}|) \times (|E_{u_i}| + |E_{v_j}|)$. Finally, the Munkres's algorithm is applied on $\mathbf{Ce}$ to find the minimum sum of edge operation costs [15], or $c_{i,j} \leftarrow c(u_i \rightarrow v_j) + \text{Munkres}(\mathbf{Ce})$.
- For node deletion, deleting a node u_i will remove all its adjacent edges, or $c_{i,\varepsilon} \leftarrow c(u_i \rightarrow \varepsilon) + \sum_{e \in E_{u_i}} c(e \rightarrow \varepsilon)$.
- For node insertion, inserting a node v_j will insert all its adjacent edges, or $c_{\varepsilon,j} \leftarrow c(\varepsilon \rightarrow v_j) + \sum_{e \in E_{v_j}} c(\varepsilon \rightarrow e)$.
- For node merging $S_{mer} \rightarrow v$, two sets of incident edges to nodes in S_{mer} and v are computed first, denoted $E_{S_{mer}}$ and E_v, respectively. Let E_{loop} be the set of edges connecting the nodes $u_i \in S_{mer}$, we introduce $E'_{S_{mer}} = E_{S_{mer}} \setminus E_{loop}$.

Then, an edge cost matrix $\mathbf{Ce}$ for two sets $E'_{S_{mer}}$ and E_v is built similarly as for node substitution. The Munkres's algorithm is also used to find the minimum cost for $E'_{S_{mer}}$ and E_v. Finally, the total cost for node merging is $c_{S_{mer},v} \leftarrow c(S_{mer} \rightarrow v) + \text{Munkres}(\mathbf{Ce}) + \sum_{e \in E_{loop}} c(e \rightarrow \varepsilon)$.
- For node splitting $u_i \rightarrow S_{spl}$, the computational steps of edge cost are similar to node merging: $c_{u,S_{spl}} \leftarrow c(u \rightarrow S_{spl}) + \text{Munkres}(\mathbf{Ce}) + \sum_{e \in E_{loop}} c(\varepsilon \rightarrow e)$.

3.3 Illustrative Example

Here we give an example of the cost computation both for node and edge operations. Let G_1 and G_2 be 2 unlabelled graphs as in Fig. 1, and each node can be merged or split to maximum 2 other nodes ($k = 2$). Suppose that nodes with an edge between them can be considered as candidates for merging/spliting. Other words, $P^k_{mer} = \{ab, bc\}$, and $P^k_{spl} = \{AB, AC, BC, CD\}$ are sets of merging and splitting nodes in G_1 and G_2, respectively. We have $n = |V_1| = 3$ and $m = |V_2| = 4$. All cost functions are defined as in Table 2.

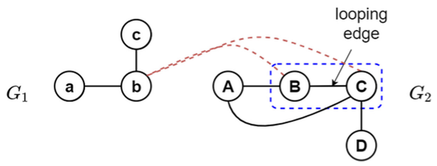


Fig. 1. The example graph with a partial matching $\lambda = \{b \rightarrow B, C\}$.

Table 2. Cost functions for edit operations in the example.

Operation	Node operation cost	Edge operation cost
Substitution	$c(u \rightarrow v) = 1$	$c(e_1 \rightarrow e_2) = 1$
Insertion	$c(\epsilon \rightarrow v) = 2$	$c(\epsilon \rightarrow e_2) = 2$
Deletion	$c(u \rightarrow \epsilon) = 10$	$c(e_1 \rightarrow \epsilon) = 10$
Merging	$c(S_{mer} \rightarrow v) = 10$	$c(e \rightarrow \epsilon) = 10, e \in E_{loop}$
Spliting	$c(u \rightarrow S_{spl}) = 1$	$c(\epsilon \rightarrow e) = 0, e \in E_{loop}$

With the partial matching $\lambda = \{b \rightarrow B, C\}$ in Fig. 1, we need to compute the sets of incidents edges to the involved nodes as follows.

- Incident edges to b: $E_b = \{(a,b), (b,c)\}$
- Looping edges: $E_{loop} = \{(B,C)\}$
- Incident edges to B, C: $E_{BC} = \{(A,B), (A,C), (B,C), (C,D)\}$
- Incident edges to B, C, except E_{loop}: $E'_{BC} = \{(A,B), (A,C), (C,D)\}$

Then, the edge cost matrix **Ce** for E_b and E_{BC} is computed in Eq. (3.3). The total cost for the partial matching $\lambda = \{b \to B, C\}$ is:

$$c_{b,BC} = c(b \to B, C) + \text{Munkres}(\mathbf{Ce}) + c(\varepsilon \to BC) = 1 + 4 + 0 = 5.$$

$$\mathbf{Ce} = \begin{array}{c|ccc|cc} & AB & AC & CD & \varepsilon_{ab} & \varepsilon_{bc} \\ \hline ab & c_{ab,AB} & c_{ab,AC} & c_{ab,CD} & c_{ab,\varepsilon} & \infty \\ bc & c_{bc,AB} & c_{bc,AC} & c_{bc,CD} & \infty & c_{bc,\varepsilon} \\ \hline \varepsilon_{AB} & c_{\varepsilon,AB} & \infty & \infty & 0 & 0 \\ \varepsilon_{AC} & \infty & c_{\varepsilon,AC} & \infty & 0 & 0 \\ \varepsilon_{CD} & \infty & \infty & c_{\varepsilon,CD} & 0 & 0 \end{array} = \begin{array}{ccc|cc} AB & AC & CD & \varepsilon_{ab} & \varepsilon_{bc} \\ \hline 1 & 1 & 1 & 10 & \infty \\ 1 & 1 & 1 & \infty & 10 \\ \hline 2 & \infty & \infty & 0 & 0 \\ \infty & 2 & \infty & 0 & 0 \\ \infty & \infty & 2 & 0 & 0 \end{array} \qquad (2)$$

4 MMAS for ExGED

4.1 Algorithmic Scheme

In this work, we use the Max-Min Ant System (MMAS) which is a variant of Ant Colony Optimization (ACO) [21]. An ant colony is employed to generate solutions to the considered problem in a parallel manner. At each iteration, each ant constructs a complete matching based on the transition probabilities. A construction graph denoted G_{ants}, is utilized to know the possibilities for one ant. In other words, G_{ants} is the search space that shows all the possibilities of node matching between 2 graphs. In this work, G_{ants} is considered as a complete undirected graph where each node in G_{ants} represents an edit operation between nodes in G_1 and G_2. Edges in G_{ants} aim to build incrementally the best edit path. Pheromones of natural ants here correspond to probabilities that are shared among ants to build a solution. An initial pheromone value is laid on each vertex of G_{ants} and the pheromone value changes according to the performance of the best matching found and the pheromone update schemes.

4.2 Construction Graph

A construction graph G_{ants} provides all possible node matching (included extended case) between two graphs G_1 and G_2. In other word, G_{ants} is built to explore all the edit paths for the ExGED problem defined in the previous section. Figure 2 presents the construction graph built from the graphs Fig. 1. The candidates for each edit operation is the Cartesian products of node sets as in Table 3. Remark that when the possibility of an ant to move on G_{ants} depends on the current partial edit path. For instance, if the partial edit path is $\lambda = \{b \to B, C\}$, all candidates contain one of these nodes are pruned (gray nodes). That means the current ant cannot move on these positions until the edit path is complete.

Table 3. Candidates for each edit operation in the construction graph, where V_1, V_2 are sets of nodes in G_1, G_2, respectively, ε is a virtual node, P^k_{mer}, P^k_{spl} are sets of merging and splitting nodes, respectively.

Operation	Substitution	Deletion	Insertion	Merging	Splitting
Candidates	$V_1 \times V_2$	$V_1 \times \varepsilon$	$\varepsilon \times V_2$	$P^k_{mer} \times V_2$	$V_1 \times P^k_{spl}$

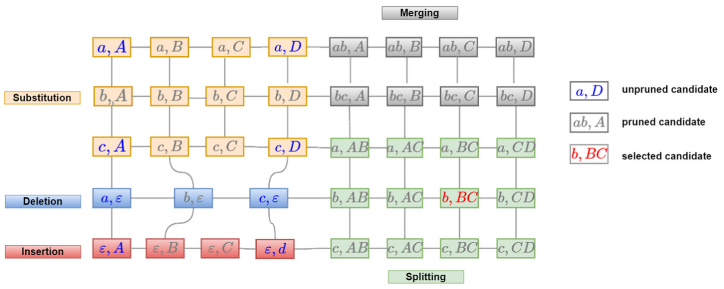

Merging
Substitution
Deletion
Insertion
Splitting
a, A a, B a, C a, D ab, A ab, B ab, C ab, D
b, A b, B b, C b, D bc, A bc, B bc, C bc, D
c, A c, B c, C c, D a, AB a, AC a, BC a, CD
a, ε b, ε c, ε b, AB b, AC b, BC b, CD
ε, A ε, B ε, C ε, d c, AB c, AC c, BC c, CD
a, D unpruned candidate
ab, A pruned candidate
b, BC selected candidate

Fig. 2. Construction graph representing all possible node matching of two graphs in Fig. 1. Some edges are not presented for readability but all nodes are fully connected.

4.3 Construction of a Complete Matching and Neighborhood Search Strategy

At each iteration, every ant builds a complete edit path. It starts with an empty edit path $\lambda = \{\}$, then it adds a new candidate $v_i \in G_{ants}$ to λ based on the transition probability, until λ contains all nodes in both graphs. The transition probability is computed based on two factors: the heuristic and the pheromone. The heuristic is derived from the cost matrix $\mathbf{C}$ and the pheromone is laid on the construction graph G_{ants}. Let τ_{v_i}, η_{v_i} be respectively the pheromone and heuristic values of the candidate v_i, then the transition probability of v_i is:

$$Pr_{v_i} = \frac{[\tau_{v_i}]^{\alpha} \times [\eta_{v_i}]^{\beta}}{\sum_{j=1}^{|cand|} [\tau_{v_j}]^{\alpha} \times [\eta_{v_j}]^{\beta}} \tag{3}$$

Besides, in this work, we enhance the method in [10] by considering a neighbor search strategy to reduce the computational time of MMAS. That means from a previously selected node in G_1, we only focus on its neighbors to find the next candidates. If no neighbor is found, a random choice is made. Due to this local strategy search, it may miss good solutions but it is less time-consuming. Admit that in the case of a complete graph, this strategy is useless because all nodes have the same number of neighbors. A demonstration for the improvement is given in Fig. 3.

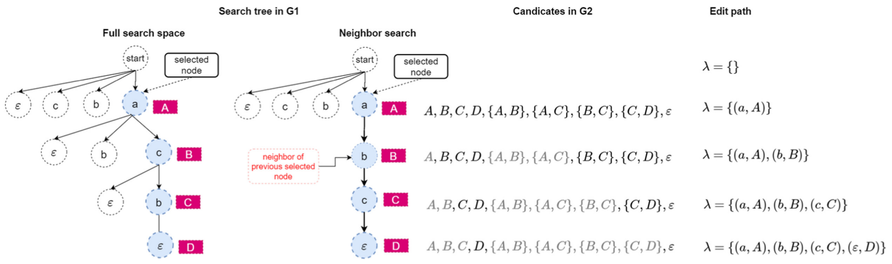

Search tree in G1
Full search space
Neighbor search
Candicates in G2
Edit path
start
selected node
neighbor of previous selected node
$A, B, C, D, \{A, B\}, \{A, C\}, \{B, C\}, \{C, D\}, \varepsilon$
$\lambda = \{\}$
$\lambda = \{(a, A)\}$
$\lambda = \{(a, A), (b, B)\}$
$\lambda = \{(a, A), (b, B), (c, C)\}$
$\lambda = \{(a, A), (b, B), (c, C), (\varepsilon, D)\}$

Fig. 3. Difference of search tree in [10] and the one with neighborhood search strategy in our work. The gray nodes indicate pruned nodes in G_2 according to the partial solution built.

4.4 Pheromone Update

The pheromone updating process happens on each node in G_{ants} after all ants have built their solution at the current iteration, including reinforcement and evaporation. The reinforcement is done only on the best solution of the current iteration (λ_{itbest}). The evaporation is done for all nodes according to the evaporation rate $\rho \in [0, 1]$. Then, the pheromone values are constrained in the interval $[\tau_{min}, \tau_{max}]$ [21].

$$\tau_{v_i} = (1 - \rho) * \tau_{v_i} + \Delta_{v_i}, \quad \Delta_{v_i} = \begin{cases} \frac{1}{1+c(\lambda_{itbest})} & \text{if} \quad v_i \in \lambda_{itbest} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

5 Experiments

Here the MMAS algorithm is used to seek the best matching between two graphs. This is because the search space is huge and the underlying combinatorial problem can not be solved exactly in a reasonable time [2]. However, one needs to tune the parameters of MMAS to be well adapted to the problem. All experiments are implemented in Python 3.8.5 and run on Windows 10 Intel(R) Core(TM) i7-8750 CPU @ 2.20 GHz, RAM of 16.0 Go. From now on, the proposed method in this work is called im-MMAS-ExGED.

5.1 Data Set and Graph Representation

For the experiments, we use the models in the SESYD dataset which contains architectural and electrical symbols[1]. The deformed symbols are created by adding noises to break one line into several lines or rotating or scaling the symbols. This new dataset representing noisy symbols is publicly available as in following link.

[1] Can be accessed here: http://mathieu.delalandre.free.fr/projects/sesyd/.

For the graph representation, nodes are lines and edges are relations between lines. Node's attributes are the relative length (l) which is normalized according to the longest line in each graph. Edge's attributes are the normalized relative angle ($\theta \in [0, 1]$) and type of relation (rel). The relation rel includes T-Junction (T), Parallel (P), Successive (S), L-Junction (L) and intersection (X) [14]. We give an example in Fig. 4.

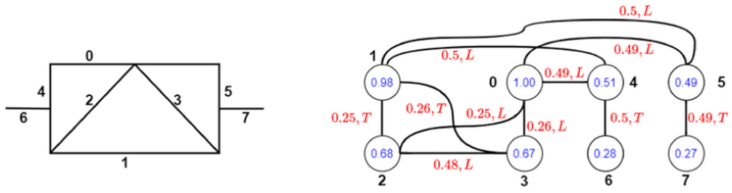


Fig. 4. The symbol (left) and its graph representation (right) where numerical attributes of nodes (blue) and edges (red) are normalized. (Color figure online)

From the observation, we suppose that the S-relation appears when there are noises. So, we set nodes that are connected by edges with S relations will be candidates for merging or splitting. Also, $k = 3$ is set according to the dataset (Fig. 5).

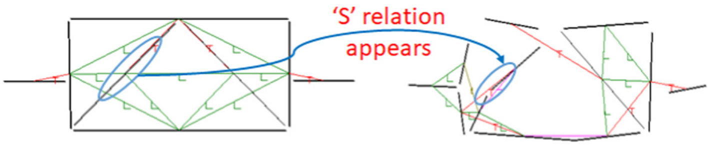


Fig. 5. The S relation appears when lines in the model (left) are split into several lines in the deformed symbol (right).

5.2 Definition of Cost Functions for Edit Operations

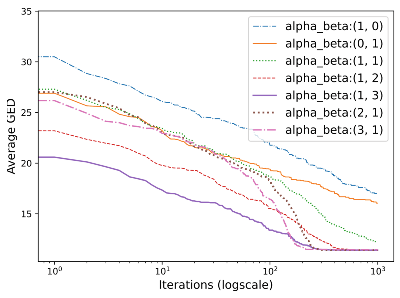
Normally, the proposal of cost functions is based on the graph knowledge, such as the node/edge's attributes and the graph's topology. Therefore, we define the cost functions for the node and edge operations vis-à -vis the previous graph representation. For node operations, the cost is proportional to the length attribute (l). That means deleting/inserting a long line should cost more than a shorter one. Substitution of two lines of equivalent length should be zero. Likewise, merging lines will be cheaper if their merging generates a line of adequate length. For edge operations, the cost depends on the relation type (rel) and the relative angle (θ). The values of edge cost functions are also in $[0, 1]$ to be compatible with the node cost functions. Details of the cost functions are in Table 4.

Table 4. Cost functions of edit operations for the data set.

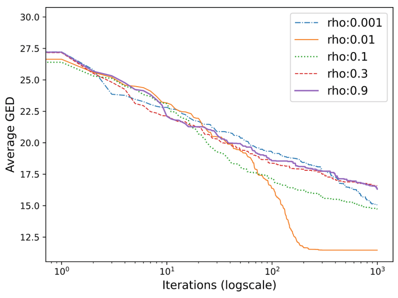
Node operation cost	Edge operation cost
$c(u \to v) = \|l_u - l_v\|$	$c(e_1 \to e_2) = \begin{cases} \|\theta_{e_1} - \theta_{e_2}\| & \text{if } rel_{e_1} = rel_{e_2} \\ 1 & \text{otherwise} \end{cases}$
$c(\varepsilon \to v) = l_v$	$c(\varepsilon \to e_2) = 1$
$c(u \to \varepsilon) = l_u$	$c(e_1 \to \varepsilon) = 1$
$c(S_{mer} \to v) = \|\sum_{u_i \in S_{mer}} l_{u_i} - l_v\|$	$c(e \to \varepsilon) = 0, e \in E_{loop}$
$c(u \to S_{spl}) = \|l_u - \sum_{v_i \in S_{spl}} l_{v_i}\|$	$c(\varepsilon \to e) = 0, e \in E_{loop}$

5.3 Parameter Setting for MMAS

In this part, we study the impacts of principal parameters of MMAS and local search on the results of GM problems. Each parameter study is run 30 times and the average results are presented. Figure 6 shows the influence of parameters α, β, ρ to the final costs. Based on the results, we chose $\alpha = 1, \beta = 3, \rho = 0.01$ and 3-opt local search for later experiments. For other parameters, we also do the experiments but they do not affect the results. We set $nb_{ants} = 5$, $[\tau_{min}, \tau_{max}] = [0.1, 2.0]$ for later experiments.



(a) α, β



(b) ρ

Fig. 6. Influences of principal parameters to the convergence of MMAS to ExGED (with 5 ants, 300 iterations, $[\tau_{min}, \tau_{max}] = [0.1, 2.0]$).

5.4 Matching Quality Analysis: Interest of Using Merging and Splitting

In this section, we evaluate the matching quality regarding the sub-part correspondence or node-to-node matching. The result of the approach is compared to one of the bipartite methods for GED [15] (denoted BP-GED) and the one in [10] (denoted old-ExGED-MMAS). We regulate a matching is reasonable if it matches correctly the split lines in the noisy symbols with its origin line in the model. The node correspondence is checked manually. We take 5 models and 6 levels of distortions for each model for this experiment.

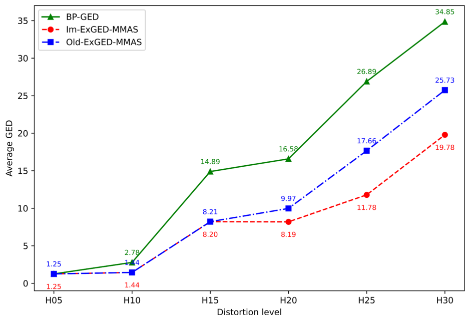


Fig. 7. Average costs of BP-GED, im-MMAS-ExGED and old-MMAS-ExGED in terms of distortion levels.

Figure 7 shows the average cost given by three methods in terms of distortion levels. Note that the cost is proportional to the distortion level in general, higher cost for higher distortion. In all cases, the BP-GED obtains a higher dissimilarity than the ExGED-based two methods. It confirms the appropriateness of the ExGED for this kind of distortion against the GED. Looking into the sub-part correspondence, the cost differences of methods are proportional with the achieved mappings. Specifically, ExGED-based methods can find more appropriate matching than BP-GED thanks to merging and splitting. A demonstration is given in Fig. 8 and Table 5. Also, the results given by the im-MMAS-ExGED are pretty competitive with the old-MMAS-ExGED. Their costs are not very bias but the im-MMAS-ExGED is much more time-saving than the old-MMAS-ExGED. For instance, the old-MMAS-ExGED needs 705.048 s to produce the result, meanwhile the im-MMAS-ExGED only needs 149.425 s with $maxiter = 500$ in this experiment. This consolidates that the proposed method is less time-consuming but still gives a good result compared with the old-MMAS-ExGED.

Table 5. Node correspondence at distorted positions between the model 032 and its noisy versions in Fig. 8 given by BP-GED and ExGED-based methods.

Methods	Level 5	Level 10	Level 15
BP-GED	$\varepsilon \rightarrow 9$	$1 \rightarrow 5; \varepsilon \rightarrow 9$	$2 \rightarrow 14; 3 \rightarrow 17; 0 \rightarrow 16; 6 \rightarrow 18$
ExGED-based	$\varepsilon \rightarrow 8$	$1 \rightarrow \{5, 9\}$	$2 \rightarrow \{14, 17\}; 4 \rightarrow \{16, 18\}$

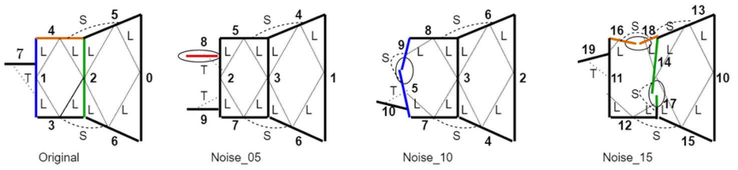


Fig. 8. Drawing of nodes (in bold) and edges (dashed lines) on the symbol 032 and its distortion levels. Ellipse indicates noise position, same noise position illustrated with the same color. (Color figure online)

5.5 Classification Problem

In the above section, we point out the interest of using merging and splitting for well recognizing the noisy symbols in terms of matching quality analysis for a small dataset. In this part, we try to indicate the efficiency of the proposed method for a bigger dataset in a classification problem. To serve this purpose, we design the experiment as follows. We select randomly 20 models in the symbol dataset as the training set (Fig. 9) and around 20 noise levels of each model as the testing set. As a consequence, the size of the training set ($\#train = 20$) is much smaller than the testing set ($\#test = 418$). We expect that the proposed method can get a good classification rate with such small training information. We use one-nearest neighbor classifier or KNN with $K = 1$ to do this task. So, the cost induced by GED-based or ExGED-based methods are utilized directly for the classification. The performance of the im-ExGED-MMAS is measured against these of BP-GED [15], Greedy algorithm for ExGED (Greedy-ExGED), old-ExGED-MMAS [10], and GED-MMAS. The parameter setting for this experiment is described in Table 6.

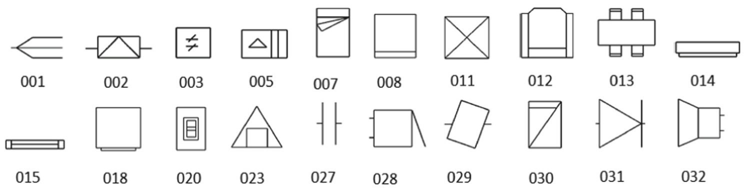


Fig. 9. Models for classification problem in the experiment.

Table 6. Parameter setting for symbol recognition problem.

Parameter	(α, β)	$maxiter$	nb_{ants}	ρ	(τ_{min}, τ_{max})	k
Value	(1,3)	50,100	5	0.01	(0.01,2.0)	2,3

The results are shown in Table 7. “Accuracy” is percentage of the number of testing symbols correctly classified. $\#matchings$ is the total number of matchings done for each method. For BP-GED and Greedy-ExGED, $\#matchings = \#test \times \#train$. For the remaining methods, $\#matchings = \#test \times \#train \times nb_{ants} \times maxiter$. “Time/Matching(s)” is the ration between the total computational time and the total number of matchings. Through Table 7, we observe the following main points. Firstly, the MMAS-based approaches give a better accuracy but more time-consuming than the other approaches (BP-GED, Greedy-ExGED). It is natural because the MMAS needs to repeat the searching process several times to improve its solution quality. Nevertheless, examining the average time for constructing a matching, the MMAS-based approaches are more time-saving. Secondly, between GED-based and ExGED-based approaches, the ExGED-based ones are usually more robust with a higher classification rate, especially the im-ExGED-MMAS. Thirdly, better results are obtained with $k = 3$ rather than $k = 2$ since it provides extra matching possibilities. In contrast, it demands more computational time due to the size increase of the cost matrix. Finally, the im-ExGED-MMAS proves its effectiveness against the old-ExGED-MMAS, which supports our initial expectation of balancing between solution quality and computational time.

Table 7. Symbol classification results given by GED-based and ExGED-based approaches, the best result of each method is in bold.

Solver	$(k, maxiter)$	Accuracy (%)	Time/matching (s)	Total time (s)	$\#matchings$
BP-GED		**83.014**	0.008	66.621	8360
Greedy-ExGED	(2,1)	74.402	0.120	96.347	8360
	(3,1)	**74.880**	0.012	100.561	8360
Im-ExGED-MMAS	(2,50)	84.450	0.002	4130.574	2090000
	(3,50)	85.885	0.002	4326.690	2090000
	(2,100)	84.211	0.002	8401.295	4180000
	(3,100)	**86.124**	0.002	8267.924	4180000
Old-ExGED-MMAS	(2,50)	84.211	0.005	9996.545	2090000
	(3,50)	85.167	0.005	10271.611	2090000
	(2,100)	83.732	0.005	20481.097	4180000
	(3,100)	**85.407**	0.005	20754.161	4180000
GED-MMAS	(_,50)	**83.253**	0.004	9451.112	2090000
	(_,100)	83.253	0.004	18683.444	4180000

6 Conclusion

This paper proposes an enhancement made to the performance of the proposed method MMAS-ExGED in [10] for the multivalent GM problem. More precisely, a neighborhood searching strategy is applied during the solution building process of MMAS to reduce the computational time. In other words, this strategy focuses only on the neighbor of the previously selected nodes to seek the next candidates. The efficiency of the proposal is shown in an application of symbol recognition at various levels of distortions. The numerical results demonstrate both qualitative (quality and interpretability of the generated matchings) and quantitative (classification performance) aspects. It confirms the advantage of using splitting and merging for symbols at multiple noise levels rather than the traditional GED-based methods. Also, it consolidates our hypothesis that the enhancement can keep the balance between the solution quality and the execution time. These optimist results encourage us to apply the proposed method to other datasets in the upcoming time, for instance, brain connectome comparison at different scales.

References

1. Ambauen, R., Fischer, S., Bunke, H.: Graph edit distance with node splitting and merging, and its application to diatom identification. In: GbRPR, pp. 95–106 (2003)
2. Berretti, S., Bimbo, A.D., Pala, P.: Graph edit distance for active graph matching in content based retrieval applications. Open Artif. Intell. J. **1**(1) (2007)
3. Berretti, S., Del Bimbo, A., Pala, P.: A graph edit distance based on node merging. In: ICIVR, pp. 464–472 (2004)
4. Boeres, M.C., Ribeiro, C.C., Bloch, I.: A randomized heuristic for scene recognition by graph matching. In: International Workshop on Experimental and Efficient Algorithms, pp. 100–113 (2004)
5. Bunke, H., Riesen, K.: Recent advances in graph-based pattern recognition with applications in document analysis. Pattern Recogn. **44**(5), 1057–1067 (2011)
6. Conte, D., Foggia, P., Sansone, C., Vento, M.: Graph matching applications in pattern recognition and image processing. In: IEEE ICIP, vol. 2, pp. II-21 (2003)
7. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. IJPRAI **18**(03), 265–298 (2004)
8. Fischer, S., Gilomen, K., Bunke, H.: Identification of diatoms by grid graph matching. In: S+SSPR, pp. 94–103 (2002)
9. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE SMC **4**(2), 100–107 (1968)
10. Ho, K.D., Ramel, J., Monmarché, N.: Multivalent graph matching problem solved by max-min ant system. In: S+SSPR, pp. 227–237 (2020)
11. Lades, M., et al.: Distortion invariant object recognition in the dynamic link architecture. IEEE Trans. Comput. **42**(3), 300–311 (1993)
12. Neuhaus, M., Riesen, K., Bunke, H.: Fast suboptimal algorithms for the computation of graph edit distance. In: S+SSPR, pp. 163–172 (2006)
13. Noma, A., Pardo, A., Cesar, R.M., Jr.: Structural matching of 2D electrophoresis gels using deformed graphs. Pattern Recogn. Lett. **32**(1), 3–11 (2011)
14. Qureshi, R.J., Ramel, J.Y., Cardot, H., Mukherji, P.: Combination of symbolic and statistical features for symbols recognition. In: ICSPCN, pp. 477–482 (2007)
15. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vis. Comput. **27**(7), 950–959 (2009)
16. Sammoud, O., Solnon, C., Ghédira, K.: Ant algorithm for the graph matching problem. In: EvoCOP, pp. 213–223 (2005)
17. Sammoud, O., Sorlin, S., Solnon, C., Ghédira, K.: A comparative study of ant colony optimization and reactive search for graph matching problems. In: EvoCOP, pp. 234–246 (2006)

18. Shokoufandeh, A., Dickinson, S.: A unified framework for indexing and matching hierarchical shape structures. In: IWVF, pp. 67–84 (2001)
19. Sorlin, S., Solnon, C., Jolion, J.M.: A generic graph distance measure based on multivalent matchings. In: Kandel, A., Bunke, H., Last, M. (eds.) Applied Graph Theory in Computer Vision and Pattern Recognition, pp. 151–181. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-68020-8_6
20. Stauffer, M., Fischer, A., Riesen, K.: Filters for graph-based keyword spotting in historical handwritten documents. Pattern Recogn. Lett. **134**, 125–134 (2020)
21. Stützle, T., Hoos, H.H.: Max-min ant system. Future Gener. Comput. Syst. **16**(8), 889–914 (2000)
22. Tefas, A., Kotropoulos, C., Pitas, I.: Using support vector machines to enhance the performance of elastic graph matching for frontal face authentication. IEEE PAMI **23**(7), 735–746 (2001)
23. Zaslavskiy, M., Bach, F., Vert, J.P.: A path following algorithm for the graph matching problem. IEEE PAMI **31**(12), 2227–2242 (2008)