UNIVERSITÉ DE TOURS

ÉCOLE DOCTORALE : MIPTIS

Laboratoire d'Informatique Fondamentale

et Appliquée de Tours (EA 6300)

THÈSE présentée par :

Kieu Diem HO

soutenue le : **04 juillet 2022**

pour obtenir le grade de : **Docteur de l'Université de Tours**

Discipline/ Spécialité : Informatique

Comparaison de Données Structurelles à l'Aide d'Algorithmes Evolutionnaires

Structural Data Comparison using Evolutionary Algorithms

THÈSE dirigée par :

M. RAMEL Jean-Yves	Professeur, Université de Tours

Co-encadrée par :

M. MONMARCHE Nicolas	Maître de Conférences, Université de Tours

RAPPORTEURS :

Mme. LUTTON Evelyne	Directrice de Recherche, INRAE-AgroParisTech
Mme. VINCENT Nicole	Professeure, Université de Paris

JURY :

M. LLADOS Josep	Professeur associé, Université Autonome de Barcelone
Mme. LUTTON Evelyne	Directeur de Recherche, INRAE-AgroParisTech
M. MONMARCHE Nicolas	Maître de Conférences, Université de Tours
M. RAMEL Jean-Yves	Professeur, Université de Tours
M. SIARRY Patrick	Professeur, Université Paris-Est Créteil Val de Marne
Mme. VINCENT Nicole	Professeure, Université de Paris

Acknowledgments

In our Vietnamese language, especially at the Buddha's door, there is a famous expression: "Life is impermanent". This expression implies that everything is moving, changing every second, just like the saying "No man ever steps in the same river twice, for it's not the same river and he's not the same man." (Heraclitus). My PhD study was the same, just over 3 years ago, I just rushed to France for my PhD and didn't know any French, now I'm about to become a PhD and can communicate fluently in French. Life is always changing, we are all evolving and moving constantly.

To achieve those developments, it requires constant effort and external energizing factors. Speaking of that, I would like to thank my supervisors, Jean-Yves RAMEL and Nicolas MONMARCHE, for giving me valuable lessons and experiences like today. I wouldn't be where I am today without the support from my supervisors. Whenever I needed them, they were always by my side, both academically and in life. Talking about Jean-Yves, I sincerely thank you for not only imparting valuable research experiences but also showing me the beauty of Graph Matching. His strict but dedicated character showed me the beauty of a true researcher. About Nicolas, I thank you for always encouraging me mentally. He always knows how to make people comfortable around him. Both of you are a great combination for me during my thesis, academically and mentally. I really appreciate the opportunity to work with you two, especially during a difficult time due to the Covid. I also thank Mostafa-'my third supervisor'. Your knowledge and your kindness help me a lot to during my research.

It would be remiss if I didn't thank the members of my PhD committee: Nicole VINCENT, Evelyne LUTTON, Patrick SIARRY and Josep LLADOS, for taking the time for my thesis defense. Especially Nicole and Evelyne have accepted to be the reviewers of my thesis manuscript. My thesis would not be complete without the contributions from the committee members.

I would also like to thank all the laboratory members who have always supported me over the years, especially Tifenn RAULT, Nicolas RAGOT, Hubert CARDOT, Mathieu DELALANDRE, and Romain RAVEUX. I also would like to thank the secretaries who always help me wholeheartedly in administrative procedures, also the IT staffs and the kind security guard Gérald.

Besides, I would like to sincerely thank the friends who have been with me during this time. My life would be so boring without you. Studying for a PhD is an opportunity for me to get to know international friends and learn about cultural and culinary diversity (Meya, Laura, David, Guillaume, Romain). Good memories of you are always in my heart.

Special thanks to my Vietnamese friends: Huyen, Hao, Tu, Hoang, Hung, Vinh, Phat, Nhi and the family of Tien, Ut. You have helped ease my homesickness and motivated me in France. You are my brothers and my sisters in France.

Other than that, I'm glad to know Touraine association. Thanks to the association, I had the opportunity to improve my French and learn more about the French culture. Thank you very much for caring Vietnamese people in France. I also thank the Hugo family, my second family in France, for always standing by and encouraging me. I always remember the great moments with you: going on vacation, hiking and playing social games. In particular, I would like to thank my family: my Grandma, my Parents, my Sister, my Brother and my Niece, who have always been by my side to help me mentally. Although we are thousands of miles apart, we are always close in mind. And I also apologize to Grandpa for not being there for him while he was gone.

This particular last place for me to say thank to my sweetheart-Hugo. Thank you for always being patient with me and supporting me. Let's go together, let's continue writing memorable journeys together.

And to close, I would like to send you one of the quotes that motivates me to do science: "Logic will get you from A to Z; imagination will get you everywhere" (Albert Einstein).

Tours, France, 01/07/2022
Kieu-Diem HO

Résumé

Les graphes sont des structures de données capables de décrire des entités complexes. Ils présentent des avantages importants par opposition aux caractéristiques numériques classiquement impliquées dans l'apprentissage automatique tels que la flexibilité de leur dimensionalité et de leur composition en sous-structures. Ces propriétés, ainsi que l'interprétabilité qu'ils apportent, constituent un intérêt particulier pour de nombreuses applications comme la chémo-informatique, la bio-informatique, la vision par ordinateur, l'indexation de vidéo, la fouille de texte, l'analyse des réseaux sociaux et du Web.

En reconnaissance structurelle des formes, les objets sont alors transformés en graphes dans lesquels les sommets correspondent aux sous-parties d'un objet et les arcs représentent les relations entre les sous-parties. Le problème de la comparaison d'objets devient alors un problème de mise en correspondance de graphes. La dissimilarité entre deux objets est évaluée sur la base de l'appariement obtenu entre les sommets et les arcs des deux graphes comparés.

Dans le domaine de l'appariement de graphes, la plupart des travaux traitent de la mise en correspondance univalente qui implique qu'un sommet/arête dans un graphe ne peut être mis en correspondance qu'avec, au maximum, un seul sommet/arête du second graphe. Pourtant, cette mise en correspondance univalente n'est parfois pas assez robuste pour couvrir tous les scénarios compliqués pouvant subvenir dans des cadres applicatifs réels. La sur-segmentation ou sous-segmentation dans le traitement des images, l'occlusion/inclusion dans le suivi vidéo, les différents types de bruit dans la reconnaissance de l'écriture manuscrite constituent des exemples pour lesquels une mise en correspondance bijective n'est pas satisfaisante. Dans ces situations, un appariement multivalent serait plus adapté car il permet à un sommet/un arc dans un graphe d'être associé à plus d'un sommet/un arc dans le second graphe. En d'autres termes, l'appariement multivalent de graphes est un problème plus général que l'appariement univalent mais le passage de l'un à l'autre génèren bien évidemment, un coût de calcul plus important. Par conséquent, malgré l'intérêt que cela pourrait avoir, seules quelques études abordent le problème de l'appariement multivalent de graphes. C'est pourquoi nous avons décidé de nous focaliser sur cette problématique dans le cadre de cette thèse.

La plupart des travaux existants proposent de résoudre le problème de l'appariement multivalent de graphes en exploitant une extension de la distance d'édition entre graphes (GED) avec des opérations d'édition supplémentaires de division et de fusion. La GED est généralement choisie car elle peut être adaptée assez facilement à de nombreux types de graphes. Cependant, aucun de ces travaux ne donne une définition formelle d'une distance

d'édition multivalente. En outre, les mises en correspondance entre arcs jouent un rôle essentiel mais souvent négligé lors d'une mise en correspondance entre 2 graphes ; alors que les arcs reflètent la structure du graphe. Bien souvent, seules les opérations sur les sommets sont correctement prises en compte pendant le processus d'appariement. Il manque alors une analyse complète des mises en relation entre les arcs induites par les opérations de fusion et de division appliquées sur les sommets. D'autres approches considèrent l'appariement multivalent de graphes comme un simple 'assignment' entre les éléments de deux ensembles et se limitent aux graphes ayant uniquement des attributs symboliques. Partant de ces constats, nous fournissons d'abord une définition claire et précise de l'appariement multivalent de graphes formulée comme une extension de la distance d'édition entre graphes, appelée Extended Graph Edit Distance (ExGED).

Ensuite, les impacts des opérations d'édition additionnelles de fusion et la division des sommets sur les transformations additionnelles appliquées sur les arcs sont analysées en détail. Nous étudions toutes les transformations (opérations d'édition) possibles entre les arcs en cas de fusion et de division des sommets. Cette analyse fournit des informations utiles pour mieux traiter les appariements entre arcs lors de la conception du solveur pour l'appariement multivalent de graphes.

Parmi les techniques d'optimisation utilisées pour résoudre le problème d'appariement de graphes tolérant aux erreurs, les méta-heuristiques constituent un bon compromis entre la recherche arborescente et l'optimisation continue car elles permettent d'obtenir des solutions de qualité sous-optimales en un temps raisonnable. Dans la famille des méta-heuristiques, l'optimisation par colonies de fourmis (ACO) est l'un des algorithmes les plus robustes. Il est prouvé que ACO obtient de meilleures performances que d'autres méta-heuristiques pour des problèmes combinatoires compliqués. Par conséquent, l'ACO peut être un bon solveur pour l'ExGED. Aussi, nous proposons de construire une matrice de coûts qui constituera les données d'entrée pour l'algorithme ACO. Cette matrice fournira les informations de qualité (les coûts) des appariements entre les paires d'ensembles de sommets de deux graphes. Les coûts de transformation des arcs engendrés localement par chaque opération d'édition sur les sommets sont intégrés dans cette matrice de coût grâce à l'analyse effectuée précédemment. De plus, une stratégie d'analyse des voisinages et de recherche locale est mise en oeuvre pour accélérer les temps de calcul de l'algorithme ACO et pour améliorer la qualité de la solution trouvée.

La méthode proposée est testée sur un ensemble de données représentant des images de symboles dégradés avec différents types de bruits (rotation, mise à l'échelle, déformation). La qualité des mises en correspondance de graphes est utilisée pour réaliser une analyse qualitative des résultats produits par la méthode ExGED tandis que les mesures de dissimilarité correspondant à la distance ExGED sont utilisées lors des évaluations quantitatives. Les résultats des différentes expérimentations montrent que l'algorithme ACO correspond au meilleur solveur pour ExGED par rapport aux autres méta-heuristiques notamment pour les problèmes de classification. Les expériences menées démontrent également que l'appariement multivalent atteint une meilleure précision que l'appariement univalent aussi bien du point de vue qualitatif (qualité des mises en correspondance) que d'un point de vue quantitatif (taux de classification). Dans un cadre applicatif pouvant nécessiter des mises en correspondance multiples entre parties d'objets, un appariement multivalent est plus pertinent et explicable qu'un appariement classique.

L'appariement multivalent de graphes n'en est qu'à ses débuts. Par conséquent, cette thèse contribue à une meilleure compréhension de ce problème, tant sur le plan théorique que pratique. Les résultats prometteurs obtenus durant cette thèse encouragent à aller plus loin et démontrent notamment qu'il est aujourd'hui possible d'outrepasser les problèmes liés à l'explosion combinatoire.

Mots-clés: reconnaissance de formes, données structurelles, comparaison de graphes, appariement muiltivalent de graphes, distance d'édition sur graphe, méta-heuristique, algorithme évolutionnaire, optimisation par colonies de fourmis, classification.

Abstract

Graphs are data structures able to describe complex entities. They have the important benefits of unconstrained dimensionality, substructure emergence and interpretability, as opposed to numerical feature classically involved in machine learning. These properties are of particular interest for many applications including chemo-informatics, bio-informatics, computer vision, video indexing, text retrieval, social network and Web analysis. In Structural Pattern Recognition (SPR), objects are transformed into graphs in which vertices correspond to subparts and edges represent relationships between subparts. Then, the problem of object comparison becomes a problem of Graph Matching (GM), where the dissimilarity between two objects is evaluated based on the vertex and edge alignments.

In the domain of GM, most of the works deal with univalent matching. That means one vertex/edge in one graph can be mapped to a maximum of one vertex/edge in another. Nevertheless, one-to-one correspondence is not robust enough to cover all complicated scenarios in real-life contexts such as over-segmentation/under-segmentation in image processing, occlusion/inclusion in video tracking, various types of noise in handwriting recognition. In these situations, the multivalent GM is more suitable than the univalent GM because one vertex/edge in one graph can be associated with more than one vertex/edge in another graph. In other words, it allows many-to-one, one-to-many besides one-to-one correspondence. So, multivalent GM is more general than univalent GM. Switching from univalent to multivalent GM has of course a strong computational cost. As a consequence, just a few studies tackle the multivalent GM problem in comparison with the univalent one. Therefore, we will shed light on the multivalent GM in this thesis.

Most works solve multivalent GM as an extension of Graph Edit Distance (GED) with additional edit operations of splitting and merging. GED is selected because it can be adaptable for many types of graphs. However, none of previous works gives a formal definition of the multivalent GED. Furthermore, the edge relations play a vital role as the vertex operations during the matching process because it reflects the graph structure. Meanwhile, it lacks a comprehensive analysis of the edge relations induced by vertex merging and splitting. Other approaches consider multivalent GM as a relational mapping, but the proposed measures are limited to graphs with only symbolic attributes. From these research gaps, we first provide a clear and precise definition for the multivalent GM based on GED formulation, called Extended Graph Edit Distance (ExGED). After that, the edge relations implied by vertex merging and splitting are analyzed comprehensively. We investigate all possible edge relations when having vertex merging and splitting. This analysis provides meaningful information to better deal with edge relations when designing the solver for the multivalent GM.

Among optimization techniques for solving the error-tolerant GM, meta-heuristics are a good trade-off between the tree search and the continuous optimization because they can achieve qualitative sub-optimal solutions in a reasonable amount of time. Within the meta-heuristic family, Ant Colony Optimization (ACO) is one of the most robust algorithms. It is proven that ACO has better performance than other meta-heuristics for complicated combinatorial problems. Therefore, ACO can be a good solver for the ExGED. Specifically, we propose constructing a cost matrix as a heuristic information for ACO. This matrix presents the goodness of mappings between pairs of vertex sets from two graphs. Furthermore, the local edge information is integrated to this cost matrix thanks to the analysis of edge relations done before. Besides, the neighborhood strategy is applied to accelerate the computational time of ACO, and the Local Search (LS) is coupled with ACO to enhance the solution quality.

The proposed method is tested on a real symbol dataset with different types of noise (rotating, scaling, deforming) and under different aspects. The low-level matching, consisting of vertex-to-vertex correspondences, is used for qualitative analysis of the results. The dissimilarity, ExGED distance, is used for the quantitative evaluations. The numerical results show that ACO outperforms other meta-heuristics for the classification problem. The conducted experiments also demonstrate that multivalent GM achieves better accuracy than the univalent matching in the graph-level context as well as for the classification problem. The vertex-to-vertex correspondences obtained by the multivalent matching are more meaningful and explainable than those from univalent matching in case of possible distortions occurring on subparts of objects.

Multivalent GM is at its very early stage. Therefore, this thesis contributes to have a better understanding of multivalent GM, both theoretically and practically. The promising results obtained through the experiments in this thesis encourage us to go further in the future.

Keywords: pattern recognition, structural data, graph comparison, multivalent graph matching, extended graph edit distance, meta-heuristics, evolutionary algorithm, ant colony optimization, classification

Contents

1 Introduction **14**

I Graph matching problem and meta-heuristic methods: state-of-the-art **18**

2 From graph matching to multivalent graph matching problems **20**
2.1 Introducing graph 20
2.2 Graph matching problems 23
2.2.1 Concept and categories 23
2.2.2 Exact graph matching 23
2.2.2.1 Graph isomorphism 23
2.2.2.2 Subgraph isomorphism 24
2.2.2.3 Maximum common subgraph 25
2.2.3 Error-tolerant graph matching 26
2.2.3.1 Substitution-tolerant subgraph isomorphism 26
2.2.3.2 Error-tolerant subgraph isomorphism 27
2.2.3.3 Error-correcting graph matching 27
2.2.3.4 Graph edit distance 28
2.2.3.5 Multivalent graph matching 32
2.3 A focus on multivalent graph matching and its applications 35
2.3.1 Multivalent graph matching 35
2.3.1.1 Relational matching for multivalent GM 35
2.3.1.2 GED for multivalent GM 38
2.3.2 Examples of multivalent GM problem in different domains 39
2.4 Summary and perspectives 41

3 Optimization techniques for error-tolerant graph matching: from general approaches to meta-heuristic algorithms **44**
3.1 Synthesis of popular approaches for error-tolerant graph matching 45

3.1.1 Tree search-based approaches . . . 45
3.1.2 Continuous relaxation-based approaches . . . 48
3.1.3 Meta-heuristic based-approaches . . . 53
3.1.4 Other learning-free approaches . . . 57
3.1.4.1 Bipartite GM heuristic . . . 57
3.1.4.2 Iterative search-based approaches . . . 60
3.1.4.3 Graph embeddings . . . 60
3.1.5 Learning graph matching: a focus on the deep learning techniques . 61
3.2 A focus on Ant Colony Optimization . . . 66
3.2.1 General concept of ACO meta-heuristics . . . 66
3.2.2 Main variants of ACO . . . 68
3.2.2.1 Ant System . . . 69
3.2.2.2 Max-Min Ant System . . . 70
3.2.2.3 Ant Colony System . . . 71
3.2.3 ACO for graph matching problems . . . 72
3.2.3.1 ANT-GM for multivalent GM . . . 72
3.2.3.2 ANT-GM'06 and ANT-GM'06+LS . . . 76
3.2.3.3 Other ACO utilisation for solving GM problems . . . 78
3.3 Conclusion on the state-of-the-art . . . 80

II ACO for solving multivalent GM based on a GED formulation 83

4 From multivalent GM to Extended GED 85
4.1 Formulate multivalent GM in the form of ExGED . . . 85
4.2 Reformulation of cost matrix for ExGED . . . 86
4.2.1 A non-square cost matrix for ExGED . . . 86
4.2.2 An insight into cost matrix for ExGED . . . 88
4.2.2.1 Dealing with the cost matrix size . . . 88
4.2.2.2 Dealing with the edge relations for merging and splitting . 90
4.2.2.3 Corresponding cost functions . . . 95
4.2.3 Enrich cost matrix with estimated costs for edge operations . . . 97
4.3 Illustrative example . . . 98
4.4 Summary . . . 103

5 Solving ExGED with ACO 105
5.1 ACO's design for ExGED . . . 105
5.1.1 Construction graph: a search space for ACO . . . 106

5.1.2 The approach with a complete search space . . . 107
5.1.3 The approach with the search space reduced by neighborhood strategy 110
5.1.4 General pipeline . . . 111
5.2 Experimental setup . . . 111
5.2.1 Data description . . . 111
5.2.1.1 Dataset . . . 111
5.2.1.2 Graph representation . . . 114
5.2.1.3 Defining cost functions for building cost matrix . . . 115
5.2.1.4 Define parameter k for splitting and merging . . . 116
5.2.2 Setup for ACO . . . 117
5.2.2.1 Setting up ACO's parameters . . . 117
5.2.2.2 Influence of local search algorithms to ACO's solution quality 122
5.2.2.3 Studying the interaction between ants via the evolution of the pheromone matrix . . . 122
5.2.2.4 Demonstrating the non-deterministic property of ACO . . . 126
5.2.2.5 Comparison between popular variants of ACO . . . 127
5.2.3 Influence of the neighborhood strategy to reduce the search space . . . 130
5.3 Comparison ACO with the greedy and random search for ExGED problem . 132
5.3.1 Comparison in the GM context . . . 133
5.3.2 Comparison in the classification context . . . 134
5.4 Evaluation between multivalent and univalent GM on symbol dataset . . . 136
5.4.1 Comparison in the GM context . . . 137
5.4.2 Comparison in the classification context . . . 138
5.5 Summary . . . 140

III Conclusion **142**

6 Conclusion and Future works **144**
6.1 Conclusion . . . 144
6.2 Perspectives and future challenges . . . 147

Appendices **148**

A Ground truth of vertex-to-vertex correspondences for a subset in the symbol dataset **151**

B Munkres assignment algorithm **153**

Bibliography **156**

Chapter 1

Introduction

Besides numerical data, structural data such as graphs are ubiquitous, such as chemical modules, brain connectome or social networks. Graphs can describe the object's attributes and structure simultaneously [Riesen, 2015]. Therefore, graph-based representation is becoming well-known in Pattern Recognition (PR) due to the developments of computer capacities. To model one object by graphs, vertices are usually the components and edges are the relations between them. In addition, the labels/attributes on vertices and edges describe the objects' features. Somehow, the vertices and edges can be defined differently regarding the application and user's target. In PR, comparison between two objects is an essential task. Presenting objects as graphs turns the problem of object comparison into the problem of Graph Matching (GM), where the similarity between two objects is evaluated based on the vertex and edge alignments [Riesen, 2015].

Over the last decades, GM has been applied in several domains, such as image analysis, object recognition, brain connectome comparison [Conte et al., 2004; Ktena et al., 2016]. Therefore, GM has become a fundamental research problem in PR. GM is divided into two main categories: exact GM and inexact GM (or error-tolerant GM) [Conte et al., 2004]. The exact GM concentrates on answering whether two graphs are identical or not, so it is a decision problem. In contrast, the inexact GM permits matching graphs with different structures and labels. It tries to minimize the distance between two graphs to obtain the highest similarity, and as a result, it is an optimization problem. In real-life scenarios, inexact GM is more suitable than the exact one because of its flexibility, and we focus on this type of GM.

In error-tolerant GM, one can have univalent or multivalent GM. The univalent GM allows one vertex/edge in one graph to match with a maximum of one vertex/edge in another, and the multivalent GM permits a set of vertices/edges from one graph to associate with one vertex/edge in another. On the one hand, thanks to the matching constraints of multivalent GM, it can cover more situations in the PR such as over-segmented/under-segmented images in image segmentation and various types of noises in handwriting recognition [Deruyver et al., 2005; Park et al., 2014]. On the other hand, the matching constraints make multivalent GM become one of the most complex problem in the GM family [Abu-Aisheh, 2016]. One part comes from the combinatorial nature of the GM problem, $\mathcal{NP}-$hard, except for the graph isomorphism [Conte et al., 2004]. Another part is due

to more combinations of vertices and edges in multivalent GM than the univalent GM. All of this raises an interesting question about how to solve such a useful but complex problem. In this thesis, we shed light on studying the multivalent GM deeper theoretically and practically.

Due to a limited amount of works for multivalent GM, it is not as well studied as other types of error-tolerant GM. Multivalent GM is usually defined as an extension of Graph Edit Distance (GED) with splitting and merging. The traditional GED optimizes the dissimilarity between two graphs by applying edit operations on vertices and edges, such as substitution, deletion, and insertion. GED is famous for its flexibility to deal with different types of graphs [Conte et al., 2004; Riesen, 2015]. Nevertheless, there is no formal definition for the multivalent GM based on this approach. From these research gaps, we propose first a formal problem formulation for multivalent GM based on GED, called Extended Graph Edit Distance(ExGED).

The complexity of the multivalent GM comes not only from the immense possibilities of matching but also from the difficulty to handle at the same time the vertex and edge matchings. From the definition of ExGED, a very important parameter is the maximum number of multiple associations k. This parameter can have a significant impact on the problem size, and so the problem complexity. Through our literature review, few papers deal with or mention explicitly the edge matching. It is similar for the GED-based approaches, i.e., the edge relations induced by vertex splitting and merging are not well considered. Generally, the structure of object is presented by edges, and studying edge relations induced by extended mappings plays a vital role when doing GM. For that purpose, the second contribution is concentrated on the analysis of the important parameter k and the edge relations prompt by the vertex splitting and merging in ExGED. These analyses help us to better understand the high complexity of multivalent GM as well as to design a better solver for ExGED afterward.

The GM problem can be solved by exact approaches or approximate ones. The exact approaches give the optimal solution for the error-tolerant GM, meanwhile the approximate approaches yield sub-optimal solutions. The approximate approaches are more capable to handle large graphs, while the exact ones are often restricted to the small graphs [Conte et al., 2004]. Considering multivalent GM in this thesis, the approximate approaches are preferred because of the high complexity of the multivalent GM problem. Among the approximated approaches for solving GM, meta-heuristics are a good compromise between the tree search and the continuous relaxation. The meta-heuristics can find good solutions in a reasonable time. Moreover, meta-heuristics are well-known for tackling complex combinatorial problems such as Quadratic Assignment Problem (QAP) and the GM problem can be formulated as a QAP [Riesen, 2015]. Therefore, we believe that meta-heuristics will be a good candidate for the multivalent GM. Within this family, Ant Colony Optimization (ACO), a global search method, is selected as the main solver for the multivalent GM. ACO is one of the leading meta-heuristic for QAPs [Achary et al., 2021]. If good results are achieved by using ACO for multivalent GM, it will pave the way for the usage of other meta-heuristics, even other efficient methods. Moreover, it provides an insight about multivalent GM to better deal with it later. But how to solve the multivalent GM in the form of ExGED efficiently with ACO is a concern. That comes up to a question of how to reduce its complexity and to consume less computational resource. Furthermore,

it is about the manner to integrate other techniques to enhance the solution quality given by ACO. Resolving all of these issues make the third contribution in the thesis.

The last contribution is about the evaluation process of the proposed method for ExGED. Because of the lack of datasets or inaccessibility of them, we generate one dedicated for the multivalent GM problems. This dataset helps us to validate the utility of the multivalent GM in PR, it is whether or not more useful to use the multivalent GM than the univalent GM. For that purpose, various types of noises are added to the original dataset. Moreover, a Ground Truth of vertex mappings (GT) for a subset of the dataset is built to serve for the quality evaluation. This GT is used for justifying the vertex-to-vertex correspondence of two graphs obtained by compared methods in the graph-level context. Other quantitative evaluations are also implemented based on the dissimilarity (ExGED distance) of two graphs. The dataset is accessible online to facilitate the comparison of other solvers for multivalent GM via the following link `http://www.rfai.lifat.univ-tours.fr/PublicData/ExGED/home.html`.

To make clear the contributions aforementioned, the thesis is arranged into two main parts as follows:

I) The first part is composed of two chapters:

- Chapter 2 is about the state-of-the-art on GM problems, multivalent GM in particular. It begins with a short introduction to graph data and its essential role in SPR. Then, the GM problems are presented with general concepts, main categories, and a quick review of their sub-problems. Then, we concentrate on the multivalent GM (definition, applications, and related works). The conclusion is about the research gaps and propositions for the multivalent GM problem.
- Chapter 3 reviews the main optimization techniques for solving the GM problems. We give attention to the sub-optimal methods due to the high complexity of multivalent GM. Specifically, the methods based on tree search, continuous relaxation, meta-heuristics, and deep learning are presented. From the literature review, we shed light on the meta-heuristics for multivalent GM, especially ACO. This section is about the general concept of ACO, works using ACO for GM problems, and its main variants. Finally, we conclude this chapter with propositions for the next part.

II) The second part is divided into two chapters of contributions:

- Chapter 4 provides a formal formulation for multivalent GM in the form of GED, called ExGED. Then, we propose to build a cost matrix $\mathbf{C}$ for ExGED as a heuristic source for the solver-ACO. We analyze $\mathbf{C}$ through the matrix size and the edge relations induced by vertex merging/splitting, which give us an insight to better design the cost matrix. Finally, an example is demonstrated for calculation steps of the cost matrix $\mathbf{C}$.
- Chapter 5 presents the solver we propose for ExGED. First, we introduce the design of ACO for solving ExGED and its general pipeline. Notably, two approaches are proposed: one with complete search space and another with a

reduced search space by applying neighborhood search. Then, a series of experiments is implemented to validate the proposed approaches in various aspects. For each aspect, a specific conclusion is given regarding the results obtained.

Chapter 6 ends with global conclusions summarizing what has been achieved in the thesis. After, we discuss the difficulties of the studied problem and the limitations of the proposed approaches. Finally, we point out perspectives and possible research directions, which can help to solve the studied problem more efficiently or to be applicable in other fields.

Part I

Graph matching problem and meta-heuristic methods: state-of-the-art

Chapter 2

From graph matching to multivalent graph matching problems

Contents

2.1 Introducing graph 20
2.2 Graph matching problems 23
2.3 A focus on multivalent graph matching and its applications . . 35
2.4 Summary and perspectives 41

Abstract
This chapter concentrates on structural data called graphs. First, we describe this type of data. We then introduce the problem of graph comparison named graph matching. Next, we present the different categories of this problem. In particular, we focus on the multivalent graph matching problem. Following this decision, the last part of the chapter is devoted to the details of this subproblem and its applications.

2.1 Introducing graph

In graph theory, a graph is structural data that contains 'vertices' and 'edges' [Wilson, 1979]. In pattern recognition, graphs are typically used to represent objects in Structural Pattern Recognition (SPR). The parts of the objects are usually the vertices, and the relations between each pair of vertices are often the edges. However, the definitions of vertices and edges can be changed depending on the user's target, dataset, or problem studied. These parts or relations may have certain characteristics to describe the object. When representing graphs, they are called labels. The set of vertices V is finite. The set of edges E connects these vertices, $E \subseteq V \times V$. An edge e connecting two vertices u, v in V is denoted $e = (u, v)$ or e_{uv}.

Definition 1 (Graph) *A graph $G = (V, E)$ consists of two main elements:*

V is a set of vertices

$E \subseteq V \times V$ *is a set of edges*

A graph G can be undirected or directed. G is an *undirected* graph when the edges have no order or direction, i.e., $e = (u, v)$ is similar to $e = (v, u)$. On the other hand, G is a *directed* graph when the edge from u to v is different from v to u, i.e., $(u, v) \neq (v, u)$. When there are no attributes on either the vertices or the edges, G is called an *unlabelled* or *unattributed* graph. In this case, we do not know other information on the vertices or edges about their features, except their connections. Thus, G describes only the object's topology, i.e., the neighborhood structure defined by the edges. When there are attributes either on the vertices or on the edges, or both, G is called labeled graph. Thus, the labeled graph expresses not only the topology but also information about the represented object (shape, color, form, type of relationship, etc.). For a labeled graph, it is denoted $G = (V, E, L_V, L_E, \mu_V, \xi_E)$, where L_V, L_E, μ_V, ξ_E are related to the attributes of the vertices and edges.

Definition 2 (Labeled graph) $G = (V, E, L_V, L_E, \mu_V, \xi_E)$ *is an labeled graph defined by the 6-tuple:*

V *is a set of vertices*

$E \subseteq V \times V$ *is a set of edges*

L_V *is a set of vertex labels*

L_E *is a set of edge labels*

$\mu : V \mapsto L_V$ *is a function that maps label* $l_u \in L_V$ *to each vertex* $u \in V$

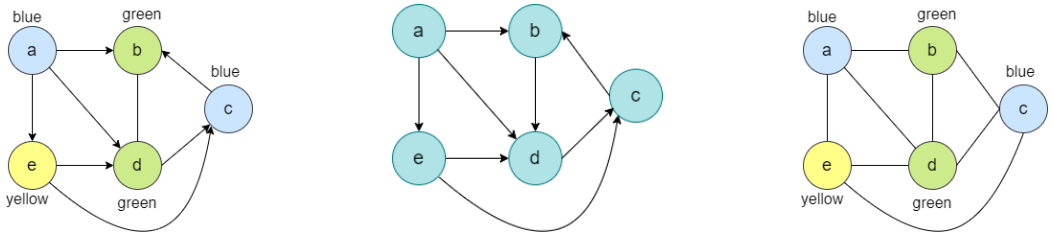
$\xi : E \mapsto L_E$ *is a function that maps label* $l_{e_{uv}} \in L_E$ *to each edge* $e_{uv} \in E$

The sets L_V and L_E can contain either symbolic or numerical attributes, or both of them. For instance, if $L_V = L'_V \cup L''_V$, where $L'_V = \{(x, y, z)\}$ is set of 3-dimensional coordinates, $L'_V = \{red, blue, \dots\}$ is set of colors. Then, L_V describes the positions and colors of vertices. Besides, $L_E = \{w_{e_{uv}}\}$ may be the set of weights that shows the importance on each edge e_{uv}. Therefore, the definition 2 permits to deal with arbitrarily structured graphs with unconstrained labelling functions. This property is essential in SPR because it allows to describe more comprehensively the represented object [Raveaux, 2019].

From here, a 4-tuple $G = (V, E, \mu_V, \xi_E)$ implies a labeled graph $G = (V, E, L_V, L_E, \mu_V, \xi_E)$ for notational convenience.

Simple graph $G = (V, E)$ is called simple graph when the set V is non-empty finite and the set E contains unordered pairs of distinct elements of V, i.e., $E = \{e = (u, v), u, v \in V, u \neq v\}$. In other words, E is finite and it does not include 'self-loop' $e = (u, u), u \in V$ [Wilson, 1979]. In this thesis, we focus on solving the simple graph.

Fig. 2.1 shows an example of three different types of graphs respecting to concepts of undirected/directed and labeled/unlabeled graphs.



(i) $G_1, e_{ab} \neq e_{ba}$ (ii) $G_2, e_{ab} \neq e_{ba}$ (iii) $G_3, e_{ab} = e_{ba}$

Figure 2.1: Examples for labeled directed graph G_1 (i), unlabeled directed graph G_2 (ii) and labeled undirected graph G_3 (iii). The set of vertex attributes for G_1 and G_3 are colors, i.e., $L_V = \{blue, green, yellow\}$.

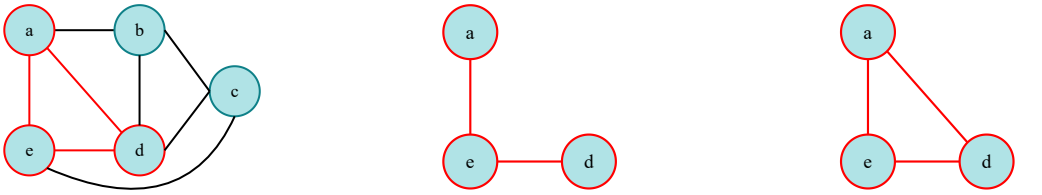
Subgraph A graph G_s is called a subgraph of G if all its vertices and edges are subsets of vertices and edges in G, respectively. In particular, a subgraph G_s of G is called induced if it preserves all the edges between the vertices in G_s that are in E of G [Wilson, 1979].

Definition 3 (Subgraph) *$G_s = \{V_s, E_s\}$ is a subgraph of $G = (V, E)$ if and only if (iff)*

$V_s \subseteq V$

$E_s \subseteq E \cap (e_{uv} \in E_s \rightarrow u, v \in V_s)$

Fig. 2.2 gives an illustration for a graph, its subgraph and its induced subgraph.



(i) $G = (V, E)$ (ii) $G_s^1 = (V_s^1, E_s^1)$ (iii) $G_s^2 = (V_s^2, E_s^2)$

Figure 2.2: Graph G (left), one of its subgraph G_s^1 (middle) and one of its induced subgraph G_s^2 (right). The subsets of vertices and edges are in red.

Graph oder The order of a graph N is equal to the number of vertices in that graph, i.e., $n = |V|$ [Harary and Norman, 1953]. In case of having two graphs G and H, we denote that $n_G = |V_G|, n_H = |V_H|$ are orders of graphs G and H, respectively. When two graphs have the same size, we denote $n = n_G = n_H$.

Graph size The size of a graph $G = (V, E)$ is the number of edges in that graph, i.e, $|E|$ [Harary and Norman, 1953].

Path A path in a graph is a sequence of vertices and edges where neither vertices nor edges are repeated.

2.2 Graph matching problems

2.2.1 Concept and categories

When dealing with numerical data like time series, we can compute distances or correlations between two elements when we want to know how similar they are. It could have the same goal with couple of graphs and one would like to compute the similarity measure between them. This kind of problem is called the graph comparison problem [Riesen, 2015].

Definition 4 (Graph comparison problem) *Given two graphs* $G = (V_G, E_G)$ *and* $H = (V_H, E_H)$*, the graph comparison problem is defined by:*

$$d : G \times H \mapsto \mathbb{R} \tag{2.1}$$

such that $d(G, H)$ presents *dissimilarity* between graphs G and H.

The similarity/dissimilarity between two graphs can be deduced from the result of the Graph Matching (GM) problem [Riesen, 2015]. GM is a process to find the correspondence between vertices and edges of two graphs. This process can follow certain constraints to ensure that substructures in the graph G are mapped to similar substructures in the graph H. Once the matching of vertices and edges is obtained, a dissimilarity score $d(G, H)$ can be deduced from these matchings and the evaluation function. In SPR, GM is classified as a structure-based recognition method, as it uses the structures of the patterns to compare them [Conte et al., 2004].

Graph matching is classified into two main categories: exact graph matching and inexact graph matching (also called error-tolerant graph matching) [Conte et al., 2004]. The first category is very strict in the sense that all topological and attribute constraints must be respected. That means two vertices/edges can be associated together if and only if they have the same label and structure. In other word, exact GM adresses the problem of detecting identical (sub)structures of two graphs and their corresponding attributes. As a result, it is well adaptable in a non-noisy context. However, these strict constraints limit its scope of application. On the other hand, the second category is more flexible because it has fewer constraints related to the topology and the attributes of the vertices/edges. In other words, it allows the matching of vertices or edges that have different labels and/or structures. Its topology can be modified during construction of the matching because it tolerates differences in structures, extra vertices or edges. Therefore, inexact graph matching is suitable for various noise and deformation contexts, which explains its applications in several fields. The details of each category are presented below.

2.2.2 Exact graph matching

2.2.2.1 Graph isomorphism

With this type of exact graph matching, the preservation of edge correspondences between the vertices of two graphs must be respected. This means that if two vertices of the first graph are connected by an edge, they must be matched with two vertices of the second graph connected by an edge, too. This condition must be respected in both

directions. Moreover, the correspondence must be bijective. It implies the one-to-one correspondence between each vertex/edge of the first graph and each vertex/edge of the second graph. In the case of labeled graphs, the labels must be identical. Two vertices are matched if and only if they have the same attributes and satisfy the edge preservation condition. The formal definition is as follows [Riesen, 2015].

Definition 5 (Graph isomorphism) *Given two labeled graphs* $G = (V_G, E_G, \mu_G, \xi_G)$ *and* $H = (V_H, E_H, \mu_H, \xi_H)$*, the graph isomorphism is a bijective mapping* $f : V_G \mapsto V_H$ *such that:*

1. *Vertex mapping preserves attributes:* $\forall u \in V_G$, $\mu_G(u) = \mu_H(f(u))$
2. *Edge-preserving:* $\forall u_i, u_j \in V_G \times V_G$, $e_{ij} = (u_i, u_j) \in E_G \iff e_{kl} = (v_k, v_l) = (f(u_i), f(u_j)) \in E_H, v_k, v_l \in V_H$
3. *Edge mapping with same attribute:* $\forall e_{ij} = (u_i, u_j) \in E_G$, $\xi_G(e_{ij}) = \xi_H((f(u_i), f(u_j)))$

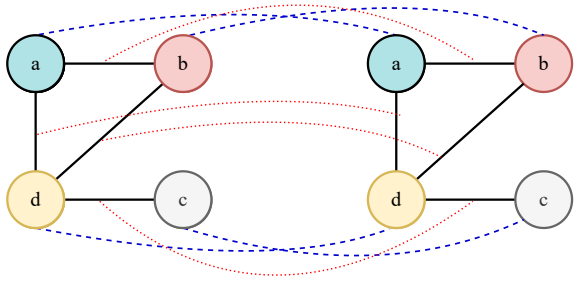


Figure 2.3: One example for graph isomorphism, blue dashed lines for vertex mappings and red dotted lines for edge mappings. vertex attributes are their colors. There is no edge attribute.

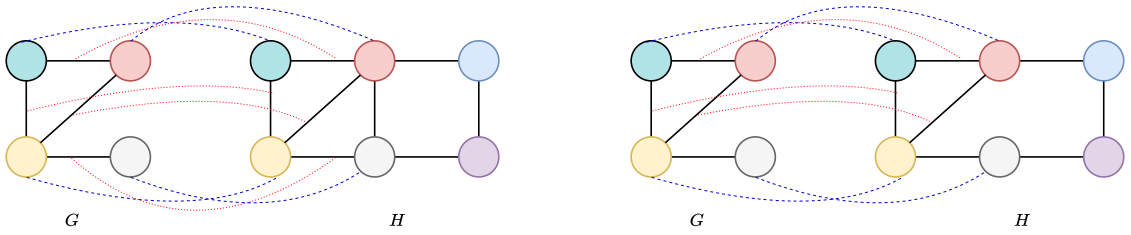
Fig. 2.3 demonstrates an example of graph isomorphism. Here, we suppose that the left and the right graphs are respective the source and target graphs. The vertex attributes are colors, and the edges are not labelled. They are also simple graphs without self-loop. According to recent studies, the complexity of this problem has not been precisely defined as $\mathcal{NP}$-complete or not. None of the algorithms can solve this problem in polynomial time. Nonetheless, the work of [Babai, 2016] proposed to tackle this problem in quasi-polynomial time inspired by methods and analysis from the group theory domain.

2.2.2.2 Subgraph isomorphism

The subgraph isomorphism is defined similarly as graph isomorphism. The only difference is that it searches for a *subgraph* G in H in the strict way. The mapping of vertices/edges must have the same attributes. Subgraph isomorphism is also called *Monomorphism.*

Definition 6 (Subgraph Isomorphism) *Given two labeled graphs* $G = (V_G, E_G, \mu_G, \xi_G,)$ *and* $H = (V_H, E_H, \mu_H, \xi_H)$*, graph isomorphism is an injective mapping* $f : V_G \mapsto V_H$ *such that:*

1. $\forall u \in V_G,\ \mu_G(u) = \mu_H(f(u))$
2. $\forall (u_i, u_j) \in V_G \times V_G,\ e_{ij} = (u_i, u_j) \in E_G,\ \exists\, e_{kl} = (v_k, v_l) = (f(u_i), f(u_j)) \in E_H, v_k, v_l \in V_H$
3. $\forall e_{ij} = (u_i, u_j) \in E_G,\ \xi_G(e_{ij}) = \xi_H((f(u_i), f(u_j)))$



(i) Subgraph isomorphism (ii) Induced subgraph isomorphism

Figure 2.4: Examples for monomorphism and induced subgraph isomorphism, blue dashed lines for vertex mappings and red dotted lines for edge mappings.

Fig. 2.4i) depicts that G is a subgraph isomorphism of H. Although there is an edge between vertices b, c in H, it is acceptable. This problem is $\mathcal{NP}$-complete as shown by [Gary and Johnson, 1979].

2.2.2.3 Maximum common subgraph

This type of problem seeks the common part of two given graphs. This part must respect the topology as well as the attributes of both graphs [Riesen, 2015]. An illustration is given in Fig. 2.5.

Definition 7 (Maximum Common Subgraph) *Given two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, G_s is the common subgraph of G and H if there exists a subgraph isomorphism from G_s to G and from G_s to H. If G_s is the largest common subgraph of G and H, then G_s is called the maximum common subgraph.*

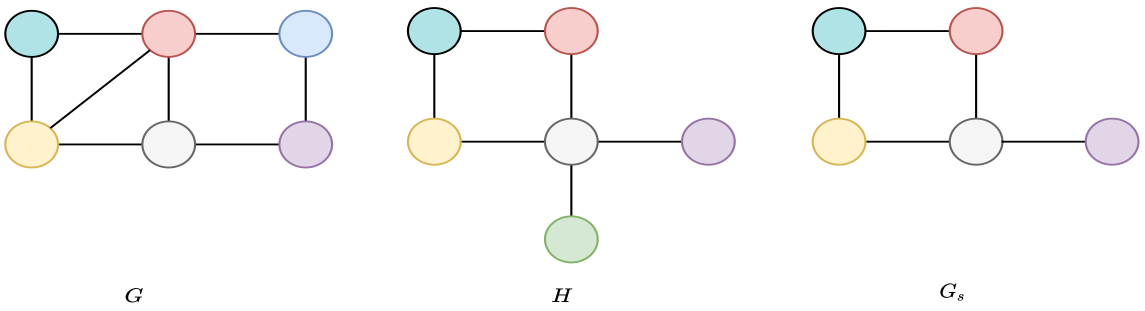


Figure 2.5: Graph G_s is a maximum common subgraph of G and H.

2.2.3 Error-tolerant graph matching

Through the exact GM problems mentioned earlier, we see that the constraints imposed for these problems are strict. To some extent, they seem too rigid for the comparison of the two graphs. It must be able to adapt to the difference between two compared graphs by relaxing or modifying the constraints that define the type of matching. Therefore, it is essential to have some tolerance in the matching process. And sometimes, one would like to calculate the similarity between two graphs rather than a boolean answer. That is the reason why we have the error-tolerant GM.

For the exact GM, one needs to verify whether the two graphs are identical or not. So, it is a decision problem. However, comparing graphs with continuous attributes is difficult to decide whether they are identical or not because matching vertices and edges between these graphs will introduce errors [Raveaux, 2019]. So, exact GM is more practical for discrete values, not continuous ones. Besides, the structures of the graphs may be distorted due to noisy data [Ambauen et al., 2003]. As a result, we can not find identical significant parts of the structure. For these situations, a better solution is associating costs or thresholds of similarity for mappings on vertices/edges, and trying to minimize these costs to achieve a good matching, not necessarily an exact matching. Hence, it is an error-tolerant GM, and the matching problem changes from a decision problem to an optimization one.

2.2.3.1 Substitution-tolerant subgraph isomorphism

With this GM problem, substitutions of vertices and edges with different attributes are permitted. Nevertheless, the topology constraints of these vertices must be followed. All the vertices and edges in the source graph need to be mapped to the vertices and edges in the target graph, although they differ in attributes. The alteration is evaluated by a distance measure called the cost of substitution. All other constraints in the subgraph isomorphism problem must be respected [Le Bodic et al., 2012].

Definition 8 (Substitution-tolerant subgraph isomorphism) *Given two labeled graphs* $G = (V_G, E_G, \mu_G, \xi_G)$ *and* $H = (V_H, E_H, \mu_H, \xi_H)$*, the substitution-tolerant subgraph isomorphism is an bijective mapping* $f : V_G \mapsto V_H$ *such that:*

1. $\forall u \in V_G,\ \mu_G(u) \cong \mu_H\left(f(u)\right)$
2. $\forall u_i, u_j \in V_G \times V_G,\ e_{ij} = (u_i, u_j) \in E_G \iff e_{kl} = (v_k, v_l) = (f(u_i), f(u_j)) \in E_H$
3. $\forall e_{ij} = (u_i, u_j) \in E_G,\ \xi_G(e_{ij}) \cong \xi_H\left((f(u_i), f(u_j))\right)$

where $\mu_G(u) \cong \mu_H(f(u))$ implies that matching two vertices are allowed even if their labels are approximately equal, and it is similar for the edge matching $\xi_G(e_{ij}) \cong \xi_H\left((f(u_i), f(u_j))\right)$.

From the definition, the only difference with the exact subgraph isomorphism problem is the equality of vertex's and edge's attributes. They can be approximate or even equal. This kind of GM can be suitable to compare graphs with numerical attributes where noises appear during the graph representation process. Fig. 2.6 gives an illustration for substitution-tolerant subgraph isomorphism.

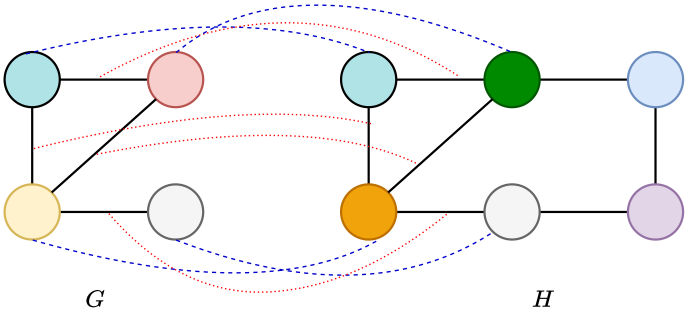


Figure 2.6: Substitution-tolerant subgraph isomorphism problem of two labeled graphs: G and H. Vertex's labels are colors. Blue dashed lines indicate vertex mappings. Red dotted lines indicate edge mappings. Mappings are edge-preserving.

2.2.3.2 Error-tolerant subgraph isomorphism

This kind of error-tolerant GM is similar to the substitution-tolerant subgraph isomorphism problem, but with fewer constraints. In detail, it permits the mappings of vertices (or edges) that have different attributes and topologies. The introduction of dummy elements (vertex/edge) helps to absord the structural modifications between two graphs. Thanks to that, each vertex/edge in graph G is mapped to a distinct vertex/edge in graph H or to a dummy vertex/edge. Depending on the optimization process, one of those decisions is taken [Messmer and Bunke, 1998].

Definition 9 (Error-tolerant subgraph isomorphism) *Given two labeled graphs* $G = (V_G, E_G, \mu_G, \xi_G)$ *and* $H = (V_H, E_H, \mu_H, \xi_H)$*, denoting* Δ_{V_H} *is a set of set of dummy vertices in* H*. The error-tolerant subgraph isomorphism is an injective mapping* $f : V_G \mapsto V_H \cup \Delta_{V_H}$ *such that:*

1. Δ_{E_H} *is a set of dummy edges*
2. $\forall u \in V_G,\ f(u) \in V_H \cup \Delta_{V_H}$
3. $\forall e_{ij} = (u_i, u_j) \in E_G,\ e_{kl} = (v_k, v_l) = (f(u_i), f(u_j)) \in E_H \cup \Delta_{E_H}$
4. $\forall u \in V_G,\ \mu_G(u) \approxeq \mu_H(f(u))$
5. $\forall e_{ij} = (u_i, u_j) \in E_G,\ \xi_G(e_{ij}) \approxeq \xi_H\left(f(u_i), f(u_j)\right)$

Fig. 2.7 depicts an example for the error-tolerant subgraph isomorphism problem. From graph G, two tolerant substitutions of vertices are done with different colors. Also, one edge deletion (black dashed line) is performed to adapt to the topology in graph H (mapping with red dashed line). This problem is also an $\mathcal{NP}$-hard minimization problem, and some suggested methods for solving it can be found in [Messmer and Bunke, 1998].

2.2.3.3 Error-correcting graph matching

For error-tolerant subgraph isomorphism, the attention is drawn on the graph G, i.e., only all vertices in graph G must be matched. For error-correcting graph matching, graphs

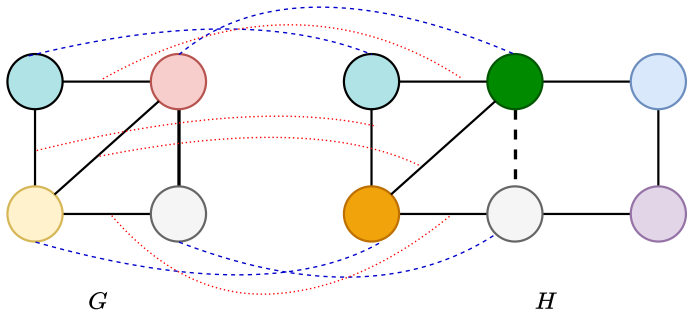


Figure 2.7: Error-tolerant subgraph isomorphism between G and H. Vertex attributes are colors. Blue dashed lines are for vertex mappings. Red dotted lines demonstrate edge mappings. The black dashed line in H represents an edge deletion.

G and H are considered equally [Raveaux, 2019]. That is done by adding dummy vertices and edges in each graph to observe the structural distorsions. In G, the set of vertices is augmented with $\mathcal{V}_\mathcal{G} = V_G \cup \Delta_{V_H}$, where Δ_{V_H} is the set of dummy vertices of G such that $|\Delta_{V_H}| = |V_H|$. Similarly, in H, $\mathcal{V}_\mathcal{H} = V_H \cup \Delta_{V_G}$, where Δ_{V_G} is the set of dummy vertices of H such that $|\Delta_{V_G}| = |V_G|$. For the edges of two graphs, it is done similarly. In G, $\mathcal{E}_\mathcal{G} = E_G \cup \Delta_{E_H}$, where $\Delta_{E_H} = |E_H|$. In H, $\mathcal{E}_\mathcal{H} = E_H \cup \Delta_{E_G}$, where $\Delta_{E_G} = |E_G|$. As a result, two graphs now have the same number of vertices ($|V_G| + |V_H|$) and number of edges ($|E_G| + |E_H|$) [Raveaux, 2019].

Definition 10 (Error-correcting graph matching) *The error-correcting graph matching is a function $f : \mathcal{V}_\mathcal{G} \mapsto \mathcal{V}_\mathcal{H}$ such that:*

1. $\forall u_i \in V_G \Rightarrow f(u_i) \in \mathcal{V}_\mathcal{H}$
2. $\forall v_i \in V_H \Rightarrow f^{-1}(v_i) \in \mathcal{V}_\mathcal{G}$
3. $\forall u_i, u_j \in V_G, e_{ij} = (u_i, u_j) \in E_G \Rightarrow (f(u_i), f(u_j)) \in \mathcal{E}_\mathcal{H}$
4. $\forall v_k, v_l \in V_H, e_{kl} = (v_k, v_l) \in E_H, \Rightarrow (f^{-1}(v_k), f^{-1}(v_l)) \in \mathcal{E}_\mathcal{G}$
5. $\forall u_i \in V_G, \mu_G(u_i) \approxeq \mu_H(f(u_i))$
6. $\forall e_{ij} = (u_i, u_j) \in E_G$, $\xi_G(e_{ij}) \approxeq \xi_H(f(u_j), f(u_j))$

Fig. 2.8 gives an illustration for this problem. Due to its flexibility, the error-correction graph matching is more general resulting in a complexity of $\mathcal{NP}$-hard [Bunke and Allermann, 1983].

2.2.3.4 Graph edit distance

The Graph Edit Distance (GED) was reported in [Bunke and Allermann, 1983; Sanfeliu and Fu, 1983]. It is a dissimilarity measure between two graphs G and H based on the optimal edit path. The edit path is a sequence of edit operations to transform one graph into another graph. Traditionally, there are three main edit operations (substitution, deletion, and insertion) that can apply on vertices and edges as presented in Table 2.1 [Riesen, 2015].

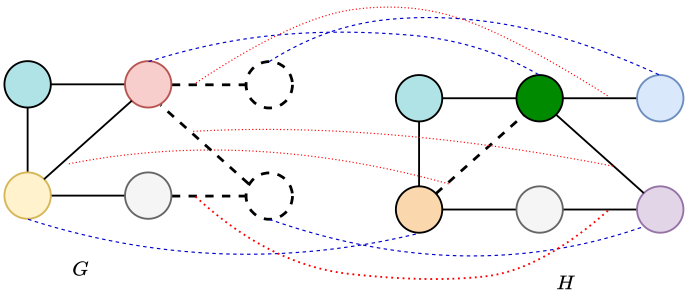


Figure 2.8: Error-correcting graph matching between G and H. Vertex labels are colors. Blue dash lines indicate vertex mappings with different labels. Red dash lines are for edge mappings with alteration. The black dashed circles and lines demonstrate dummy vertices and edges, respectively.

Definition 11 (Edit operation constraints) *When doing an operation on vertex or edge, these constraints should be satisfied:*

1. *deletion of a vertex implies deletion of all its incident edges*
2. *insertion of an edge is allowed if and only if there are two vertices that are already existing or have been inserted*
3. *insertion of an edge must not create a self-loops*

Edit operation	Vertex	Edge
Substitution	$u \to v$	$e_{ij} \to e_{kl}$
Deletion	$u \to \varepsilon$	$e_{ij} \to \varepsilon$
Insertion	$\varepsilon \to v$	$\varepsilon \to e_{kl}$

Table 2.1: Notations of classical edit operations for GED, where $u \in V_G$, $v \in V_H$, $e_{ij} \in E_G$, $e_{kl} \in E_H$, ε is a dummy vertex/edge.

Definition 12 (Edit path) *An edit path is a sequence of edit operations $\lambda = \{ed_i\}_{i=1,\ldots,k}$ that are used to transform one graph into another. A valid edit path should respect the following conditions:*

- *k is a finite positive integer*
- *for each vertex and edge, a unique operation is applied on it*

A *complete edit path* is an edit path that transforms G completely into H.

Once an edit operation is applied on vertex or edge, a corresponding cost is assigned to indicate the change strength an edit operation had done. These costs are defined by cost functions which might depend on the dataset, graph representation, and user's target. These functions are always non-negative. Therefore, the sum of costs given by edit

oeprations in GED is equal or greater than 0. There exists several possible edit paths for matching of two graphs. A value of 0 indicates a perfect similarity between two graphs, i.e., they are identical. GED tries to minimize the dissimilarity between G and H by finding the best edit path that consists of edit operations with lower cost than the others [Riesen, 2015].

Definition 13 (Graph edit distance) *Given two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, the graph edit distance between G and H is a dissimilarity measure defined by:*

$$d_{min}(G, H) = \min_{\lambda \in \Theta(G,H)} \sum_{ed_i \in \lambda} c(ed_i) \tag{2.2}$$

where ed_i is edit operation and $\Theta(G, H)$ is set of all complete edit paths between G and H.

Besides constraints on the edit operations and edit paths, there are constraints imposed on the cost functions in order to become a metric [Neuhaus and Bunke, 2007; Serratosa, 2019]. These constraints imply a priority for substitution rather than deletion and insertion on vertices and edges. That is to say, the subtitution cost should be lower than the deletion/insertion cost. In addition, the identity property also implies that $d_{GED}(G, H) = 0 \Leftrightarrow G = H$.

Definition 14 (Cost function constraints to become a metric) *Let $G = (V_G, E_G, \mu_G, \xi_G)$ and $H = (V_H, E_H, \mu_H, \xi_H)$ be two labeled graphs, to transform G into H by GED, the following conditions must be held for cost functions of GED to become a metric. Given $u \in V_G, v \in V_H, e_{ij} = (u_i, u_j) \in E_G, e_{kl} = (v_k, v_l) \in E_H$:*

1. *Non-negative: $c(\cdot) \geq 0$*
   - $c(u \to v) \geq 0$
   - $c(e_{ij} \to e_{kl}) \geq 0$
   - $c(u \to \varepsilon) > 0$
   - $c(e_{ij} \to \varepsilon) > 0$
   - $c(\varepsilon \to v) > 0$
   - $c(\varepsilon \to e_{kl}) > 0$
2. *Identity of indiscernible elements*
   - $c(u \to v) = \begin{cases} 0 & \text{if} \quad \mu_G(u) = \mu_H(v) \\ \vartheta > 0 & \text{otherwise} \end{cases}$
   - $c(e_{ij} \to e_{kl}) = \begin{cases} 0 & \text{if} \quad \xi_G(e_{ij}) = \xi_H(e_{kl}) \\ \kappa > 0 & \text{otherwise} \end{cases}$
3. *Symmetry*
   - $c(u \to v) = c(v \to u)$
   - $c(e_{ij} \to e_{kl}) = c(e_{kl} \to e_{ij})$

- $c(u \to \varepsilon) = c(\varepsilon \to u)$
- $c(\varepsilon \to v) = c(v \to \varepsilon)$
- $c(e_{ij} \to \varepsilon) = c(\varepsilon \to e_{ij})$
- $c(e_{kl} \to \varepsilon) = c(\varepsilon \to e_{kl})$

4. Triangle inequality

- $c(u \to v) \leq c(u \to \varepsilon) + c(\varepsilon \to v)$
- $c(e_{ij} \to e_{kl}) \leq c(e_{ij} \to \varepsilon) + c(\varepsilon \to e_{kl})$

Nevertheless, in a concrete application, the metric properties of GED do not always hold [Serratosa, 2019]. Solving the GED problem is a matter of minimizing the costs of edit operations to transform one graph into another. Each cost is derived from a function that can take the attributes of vertices or edges, even a constant. On the other hand, these cost functions also depend on the user's objective for learning a specific goal, i.e., objective driven cost functions. For example, it can be a comparison between the ground truth and the noisy graphs, where the Ground Truth (GT) is given by expert knowledge. Designing good functions for an application sometimes means getting the closest accuracy to the expected results. In [Serratosa, 2019], the author points out that in some cases, higher accuracy is achieved when the cost functions do not respect all the properties of the metric. Thus, one conclusion is that the constraints for the cost functions in Definition 14 are necessary for some methods to solve the GED problem but not mandatory in all cases. There are two main approaches to define cost functions: 1) manually based on a priori knowledge [Ambauen et al., 2003; Abu-Aisheh et al., 2016], 2) learning cost functions [Santacruz and Serratosa, 2020; Riba et al., 2021; Rica et al., 2021]. The second way is a problem in itself and beyond our focus in this thesis. The choice of cost functions depends on user's target and problem's information.

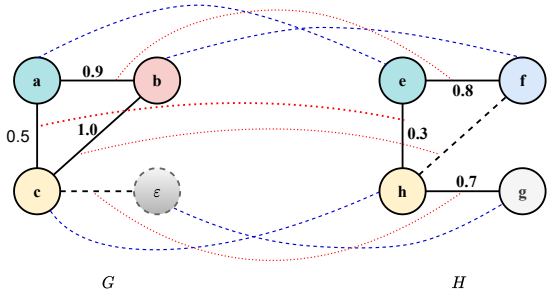


Figure 2.9: GED mapping between G and H, blue dashed lines for vertex mappings, red dashed lines for the edge mappings, ε is virtual vertex for insertion/deletion, black dashed lines for inserted/deleted edges.

Operations	Cost functions
$u \to v$	$\mathbb{1}_{\{l_u \neq l_v\}}$
$u \to \varepsilon$	1
$\varepsilon \to v$	1
$e_G \to e_H$	$\lvert l_{e_G} - l_{e_H} \rvert$
$e_G \to \varepsilon$	l_{e_G}
$\varepsilon \to e_H$	l_{e_H}

Table 2.2: One proposed cost functions for edit operations in Fig. 2.10, where $\mathbb{1}_{\{\cdot\}}$ is the characteristic function, $u \in V_G, v \in V_H$.

Fig. 2.9 demonstrates a possible transformation from G to H. Fig. 2.10 details all edit operations to get the mapping in Fig. 2.9. As we can see, each edit operation is applied uniquely on vertex or edge. Also, edge operation should be followed by the vertex operation. These operations allow adjustment step-by-step to convert G into H. After each modified step, the partial edit cost will increase as well until it becomes a complete edit

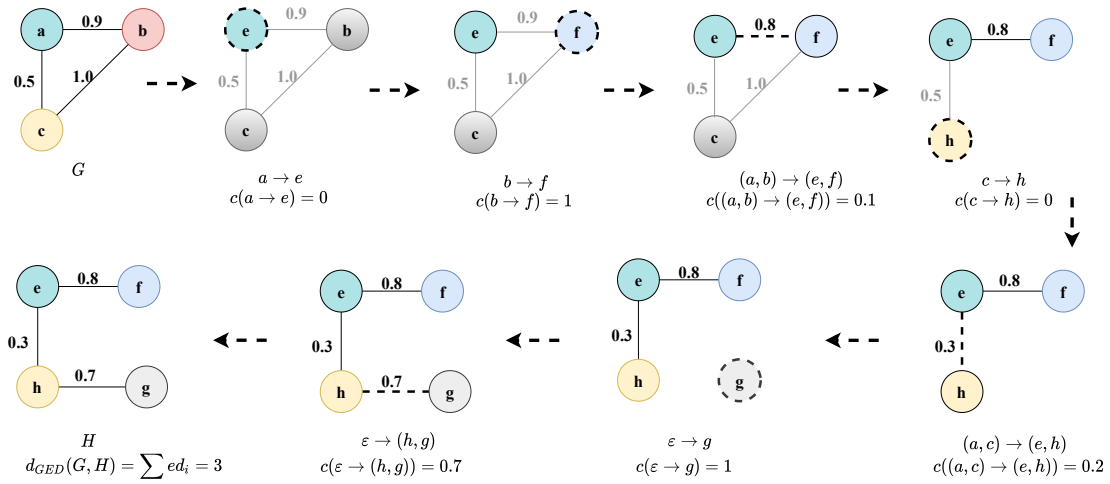


Figure 2.10: One edit path to transform G into H along with its partial costs and final dissimilarity regarding cost functions in Table 2.2. Vertex's attributes are colors and edge's attributes are normalized weights. Black dash circles/lines denote the current edit operations; grey lines/circles denote unhandled vertices/edges. The process order is indicated by arrows.

path. According to Fig. 2.10, the complete edit path in this case is $\lambda(G,H) = \{a \to e; b \to f; (a,b) \to (e,f); c \to h; (a,c) \to (e,h); \varepsilon \to g; \varepsilon \to (h,g)\}$, and the final dissimilarity between G and H regarding cost functions is $d_{GED}(G,H) = 3$.

GED shows its generality and flexibility thanks to edit operations. Besides the dissimilarity measure, GED provides the correspondences between subparts to interpret the difference at a lower level (vertex-to-vertex and edge-to-edge) [Riesen, 2015]. This feature seems to be essential in some domains where the correspondence of subparts is crucial such as in brain connectome comparison [Osmanlıoğlu et al., 2019], or keyword spotting in document image analysis [Riba et al., 2021]. That explains why GED is one of the most popular methods for error-tolerant GM [Stauffer et al., 2017; Darwiche, 2018]. Nevertheless, GED presents some challenges. Firstly, the cost functions for edit operations have significant impacts on the GED results as mentioned above. Secondly, GED is an $\mathcal{NP}$-hard problem [Riesen, 2015], so solving it efficiently in a reasonable time is another issue. Numerous heuristic methods have been studied to address this question [Riesen et al., 2007b; Serratosa, 2015; Mostafa et al., 2019]. One possible answer is to check the application demands and related factors such as graph size, types of attributes. Based on these factors, a decision can be made as to whether to use an exact or heuristic method and whether a trade-off between accuracy and speed is necessary. These challenges must be incorporated when dealing with GED problems to design better solvers [Darwiche, 2018].

2.2.3.5 Multivalent graph matching

The exact GM or error-tolerant GM presented above allow only one-to-one correspondence, i.e., one vertex/edge in one graph can be mapped to a maximum of one vertex/edge

in another. Hence, these problems belong to the family of *univalent GM*. However, this type of GM is not always sufficient in reality. That is why the multivalent GM is introduced for more general and complicated situations.

Multivalent GM problem is a kind of general GM where each vertex/edge in one graph can be mapped to more than one vertex/edge in another graph [Champin and Solnon, 2003]. It allows *one-to-many* and *many-to-one* mappings in addition to univalent mapping. This kind of GM problem is introduced to handle some special cases of image processing or applications where one-to-one mappings are not strong enough to cover all possible associations. For example, it is crucial to allow multiple mappings (one-to-many or many-to-one) to deal with complex structured objects in Computer-Aided Design (CAD) [Champin and Solnon, 2003]. According to [Sorlin et al., 2007], multivalent GM can be transformed into other kinds of matching easily with appropriate constraints. For example, if we limit the number of vertices to which a vertex can be associated to one, the multivalent GM will become the univalent GM. By this flexibility, we will consider the multivalent GM as the main studied subject in this thesis. We will detail it later in a separate section. Fig. 2.11 demonstrates a multivalent mapping of vertices and edges for two undirected labeled graphs.

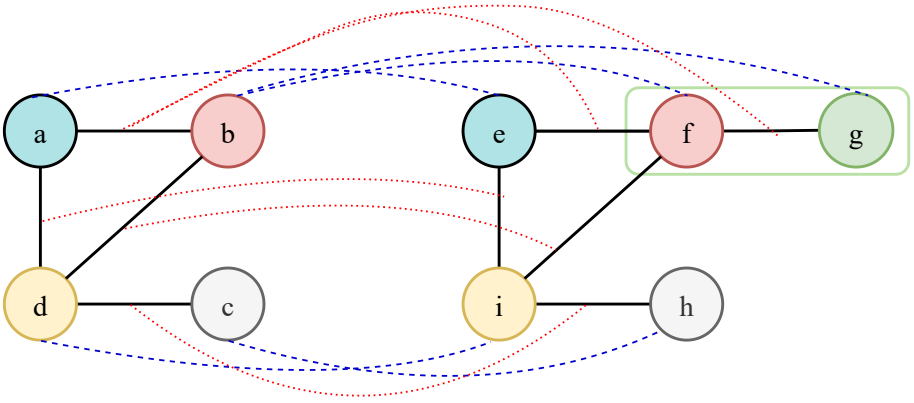


Figure 2.11: Multivalent graph matching between G and H. Vertex attributes are colors. Blue dashed lines are for vertex mappings, red dashed lines are for edge mappings. In this case, $a \to e$ and $b \to \{f, g\}$ lead to $(a, b) \to \{(e, f), (e, g)\}$

.

Apart from multivalent GM, there is many-to-many GM. That can be considered as an extended case of multivalent GM because it enables many-to-many mapping between vertices/edges. It is the most general problem where all types of tolerance are allowed: structure, labels, and number of vertices mapped. Consequently, it has higher complexity than the previous GM problems [Grapa et al., 2019].

Let Δ_{V_G} and Δ_{V_H} be two sets of dummy vertex in G and H respectively, Δ_{E_G} and Δ_{E_H} be sets of dummy edges in G and H respectively, then the extended sets of vertices and edges in each graph are:

- in $G : \mathcal{V}_\mathcal{G} = V_G \cup \Delta_{V_G}, \mathcal{E}_\mathcal{G} = E_G \cup \Delta_{E_G}$
- in $H : \mathcal{V}_\mathcal{H} = V_H \cup \Delta_{V_H}, \mathcal{E}_\mathcal{H} = E_H \cup \Delta_{E_H}$

Definition 15 (Many-to-many graph matching) *Given k is maximum number of vertices that one vertex can be matched with, the many-to-many graph matching from G to H is a mapping $f : \mathcal{V}_\mathcal{G} \mapsto \mathcal{V}_\mathcal{H}$ such that:*

1. $S_G \subset V_G, 1 < |S_G| \leq k$ *is set of vertices in* G
2. $S_H \subset V_H, 1 < |S_H| \leq k$ *is set of vertices in* H
3. $\forall u \in V_G \Rightarrow f(u) \in \mathcal{V}_\mathcal{H}, |f(u)| \leq 1$
4. $\forall v \in V_H \Rightarrow f^{-1}(v) \in \mathcal{V}_\mathcal{G}, |f^{-1}(v)| \leq 1$
5. $S_G \subset V_G \Rightarrow f(S_G) \subset V_H, 1 \leq |f(S_G)| \leq k$
6. $S_H \subset V_H \Rightarrow f^{-1}(S_H) \subset V_G, 1 \leq |f^{-1}(S_H)| \leq k$
7. $\forall e_{ij} = (u_i, u_j) \in E_G \Rightarrow (f(u_i), f(u_j)) \in \mathcal{E}_\mathcal{H}$
8. $\forall e_{kl} = (v_k, v_l) \in E_H \Rightarrow (f^{-1}(v_k), f^{-1}(v_l)) \in \mathcal{E}_\mathcal{G}$
9. $\forall u_i \in V_G, \mu_G(u_i) \approxeq \mu_H(f(u_i))$
10. $\forall e_{ij} = (u_i, u_j) \in \mathcal{E}_\mathcal{G}, \xi_G(e_{ij}) \approxeq \xi_H(f(u_i), f(u_j))$

From definition 15, many-to-many GM allows more mapping possibilities rather than the other GM problems. The 3rd and 4th statements present for univalent mappings (one-to-one, zero-to-one, one-to-zero). The 5th and 6th statements are related to multivalent mappings (many-to-one, many-to-many, one-to-many). The 7th and 8th statements guarantee that all edges are mapped together. The 9th and 10th statements ensure the mapping of attributes on vertices and edges. Concerning the edge relations, it is still one-to-one mapping or is not mentioned in the case of many-to-many according to the current related works [Grapa et al., 2019].

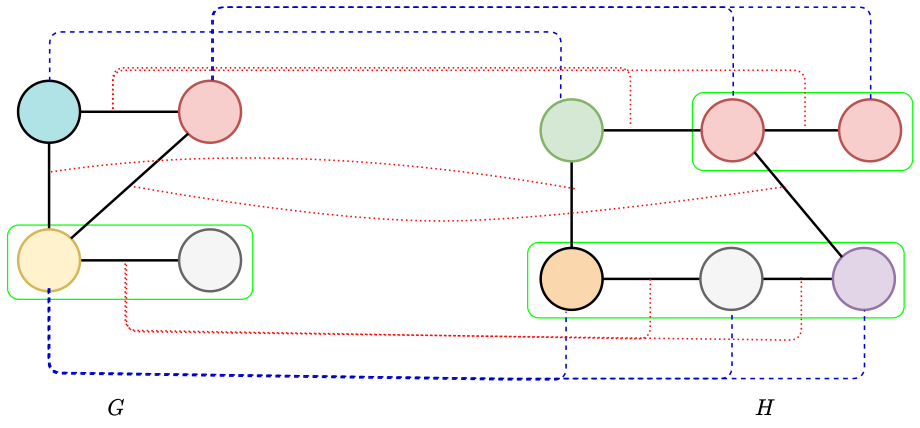


Figure 2.12: Many-to-many graph matching between G and H. The blue dashed lines represent vertex mappings (substitution-tolerant, one-to-many, many-to-many). The red dashed lines represent the edge mappings (one-to-one, one-to-many).

Figure 2.12 shows an example of many-to-many GM. As mentioned, it allows all matching possibilities (one-to-one, many-to-one, many-to-many, one-to-one, etc.). Therefore, its complexity is $\mathcal{NP}$-hard. Because of this high complexity, almost all the methods proposed to solve the problem are heuristic. In this thesis, we concentrate more on multivalent GM. It means all mappings permitted except many-to-many ones. We limit this kind of mapping due to more complicated edge relations generated after having many-to-many vertex mapping. Clarifying these edge relations is still a concern. Therefore, we aim to work with its sub-problem first, multivalent GM. From that, more theories could be developed for many-to-many GM afterward. Further details about methods for solving many-to-many GM can be referred to [Shokoufandeh et al., 2012].

2.3 A focus on multivalent graph matching and its applications

As stated previously, multivalent GM is our principle problem in the thesis. This section is dedicated to present more comprehensively the multivalent GM problem. Firstly, we recall the definitions of multivalent GM in the references. Then, we give some remarks related to the edge relations following the multivalent mapping of vertices. Finally, we review some applications of multivalent GM in different fields where our motivation for studying multivalent GM comes from.

2.3.1 Multivalent graph matching

2.3.1.1 Relational matching for multivalent GM

Multivalent GM means that each vertex in one graph can be associated with an empty set of vertices, one vertex, or several vertices in the other graph. Consequently, edge mappings are induced from these vertex mappings. The mappings of vertices and edges can be one-to-one, many-to-one, one-to-many, or one-to-empty. So, its definition is relatively similar to that of many-to-many, except for the association of two sets of vertices. Here, a formal definition of multivalent GM is taken from [Sorlin et al., 2007] which is the most approved. According to [Sorlin et al., 2007], the multivalent GM problem is also called the relational matching since GM is now a relation $m \subseteq V_G \times V_H$ rather than a function. Mathematically,

Definition 16 (Multivalent mapping of vertices) *Given two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, a multivalent matching (m) between G and H is a relation between V_G and V_H, i.e., $m \subseteq V_G \times V_H$, where $V_G \cap V_H = \emptyset$ to do not lose the generality. Specifically, given a matching m, we denote $m(u)$ the set of vertices mapped to a vertex u. More formally, a matching is defined by:*

1. $\forall u \in V_G$, $m(u) = \{v \in V_H | (u, v) \in m\}$
2. $\forall v \in V_H$, $m(v) = \{u \in V_G | (u, v) \in m\}$

When there is no constraint on the matching, i.e., each vertex may be mapped to 0, 1 or several vertices, the matching m is said to be multivalent.

Regarding the constraints on the number of vertices that a vertex can be mapped to, matching m can be partial functions, total functions, univalent matchings, injective matchings, and bijective matchings as follows [Sorlin et al., 2007]:

1. If m links all vertices in V_G to *at most one* vertex in V_H, m is then called a *partial function*
$$\forall u \in V_G, |m(u)| \in \{0, 1\}$$

2. If m links all vertices in V_G to *exactly one* vertex in V_H, m is then a *total function*
$$\forall u \in V_G, |m(u)| = 1$$

3. If m links each vertex in V_G and in V_H to *at most* one vertex, m is then a *univalent matching*
$$\forall u \in V_G, |m(u)| \leq 1 \wedge v \in V_H, |m(u)| \leq 1$$

4. If m links distinct vertices in V_G to *distinct* vertex in V_H, m is then an *injective matching*
$$\forall u \in V_G, |m(u)| = 1 \wedge v \in V_H, |m(v)| \leq 1$$

5. If m links each distinct vertices in V_G to *each distinct* vertex in V_H, m is then a *bijective matching* (one-to-one correspondence)
$$\begin{array}{rcl} \forall u \in V_G, |m(u)| = 1 & \wedge & v \in V_H, |m(v)| = 1 \\ \forall u_i, u_j \in V_G, u_j \neq u_j & \Rightarrow & m(u_i) \neq m(u_j) \\ \forall v_k, v_l \in V_H, v_k \neq v_l & \Rightarrow & m(v_k) \neq m(v_l) \end{array}$$

6. If m links each vertex in V_H to *at least one* vertex in V_G, m is then a *surjective matching* (many-to-one correspondence)
$$\forall u \in V_G,\ |m(u)| = 1 \wedge \forall v \in V_H,\ |m(v)| \geq 1$$

Following definition 16, the edge relation induced by a matching m is defined accordingly. Formally,

Definition 17 (Edges mapping induced by a vertex mapping) *Given a matching m of $G = (V_G, E_G)$ and $H = (V_H, E_H)$, an edge $(u_i, u_j) \in E_G$ to be matched with another edge $(v_k, v_l) \in E_H$ if and only if $\{(u_i, u_j), (v_i, v_j)\} \subseteq m$. By extension, $m(u_i, u_j)$ is introduced as the set of edges matched with the edge (u_i, u_j) derived from the matching m, or*

1. $\forall (u_i, u_j) \in E_G, m(u_i, u_j) = \{(v_k, v_l) \in E_H | v_k \in m(u_i), v_l \in m(u_j)\}$
2. $\forall (v_k, v_l) \in E_H, m(v_k, v_l) = \{(u_i, u_j) \in E_G | u_i \in m(v_k), u_j \in m(v_l)\}$

Definition 18 (Subgraph induced by a matching) *Given a matching m of two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, the subgraph induced by a matching m of G (resp. H) is $G_m = (V_{G_m}, E_{G_m})$ (resp. $H_m = (V_{H_m}, E_{H_m})$), where V_{G_m} and E_{G_m} (resp. V_{H_m} and E_{H_m}) are the sets of vertices and edges of G (resp. H) that are associated with at least one vertex or edge in H (resp. G). Formally,*

1. $V_{G_m} = \{u \in V_G | m(u) \neq \emptyset\}, E_{G_m} = \{(u_i, u_j) \in E_G | m(u_i, u_j) \neq \emptyset\}$
2. $V_{H_m} = \{v \in V_H | m(v) \neq \emptyset\}, E_{H_m} = \{(v_k, v_l) \in E_H | m(v_k, v_l) \neq \emptyset\}$

If the subgraph of G induced by m, $G_m = (V_{G_m}, E_{G_m})$, is equal to G, then m is an homomorphism between G and H. In other words, m is a function that associates each edge of G to an edge of H [Sorlin et al., 2007].

An illustrative example for these definitions is given in Fig. 2.13. A matching between G and H is $m = \{(1, a), (1, d), (2, b), (3, c)\}$. From matching m, we enumerate set of vertices (resp. edges) each vertex (resp. edge) matched with in Table 2.3. For example, the set $m(1)$ contains vertices matched with vertex $1 \in V_G$, i.e., a and $d \in V_H$. So, we have a multivalent mapping of vertex $1 \rightarrow (a, d)$. For other vertex mappings, they are all univalent. Specially, vertex $4 \in V_G$ is not associated to any vertex in V_H. The matching m here is a general matching. It is neither partial nor total functions because all the conditions are not satisfied. About edges matchings induced by m, when we have the multivalent matching of vertices, it is possible to have multivalent matching of edges. For instance, when $m(1) = \{a, d\}$ and $m(2) = b$, we have $m(1, 2) = \{(a, b), (b, d)\}$. That means the edge $(1, 2) \in E_G$ is mapped to (a, b) and (b, d) in E_H at the same time. It is similar for the case of $m(2, 1)$. Overally, this example shows that multivalent matching can have one-to-many, one-to-one, one-to-empty, empty-to-one, many-to-one mappings of vertices and edges as mentioned before. From mappings of vertices and edges induced from matching m, we obtain the subgraphs G_m and H_m induced from m of G and H respectively as in Fig. 2.13 b). In this case, G_m is not equal to G, m is not an homomorphism between G and H as a result.

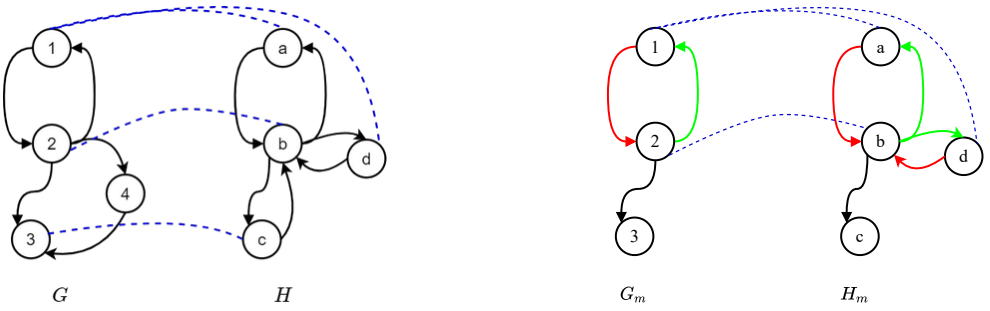


Figure 2.13: Two directed graphs with a matching $m = \{(1, a), (1, d), (2, b), (3, c)\}$ (i) and subgraphs induced by m (ii). The edges with same color indicate the same mappings. Vertex mappings are illustrated in dashed blue lines. Figure is modified from the one in [Sorlin et al., 2007].

$m(u), u \in V_G$	$m(v), v \in V_H$	$m(u_i, u_j), (u_i, u_j) \in E_G$	$m(v_k, v_l), (v_k, v_l) \in E_H$
$m(1) = \{a, d\}$	$m(a) = \{1\}$	$m(1, 2) = \{(a, b), (d, b)\}$	$m(a, b) = \{(1, 2)\}$
$m(2) = \{b\}$	$m(d) = \{1\}$	$m(2, 1) = \{(b, a), (b, d)\}$	$m(d, b) = \{(1, 2)\}$
$m(3) = \{c\}$	$m(b) = \{2\}$	$m(2, 3) = \{(b, c)\}$	$m(b, a) = \{(2, 1)\}$
$m(4) = \emptyset$	$m(c) = \{3\}$	$m(2, 4) = \emptyset$	$m(b, d) = \{(2, 1)\}$
		$m(4, 3) = \emptyset$	$m(b, c) = \{(2, 3)\}$
			$m(c, b) = \emptyset$

Table 2.3: Vertices and edges matched induced from matching $m = \{(1, a), (1, d), (2, b), (3, c)\}$ for graphs G and H in Fig. 2.13.

From Definition 16 of [Sorlin et al., 2007], we notice that the matching m expresses only vertices and edges that are associated together. The other vertices and edges that are unmatched are not in m. It can be inconvenient if there is a lot of noise in graphs where deletion and insertion operations on vertices/edges may be in demand. Champin and Solnon in [Champin and Solnon, 2003] utilized the matching m for presenting the solution of a multivalent GM problem. Nevertheless, the proposed method is applied only to graphs with symbolic attributes. The edge relation induced by vertex merging and splitting is not analyzed yet. These limitations can be overcomed if we consider the multivalent GM as en extension of GED as presented in the following.

2.3.1.2 GED for multivalent GM

Besides the mentioned definition, other works define multivalent GM as an extended GED. That is done by adding splitting or merging operations on vertices [Ambauen et al., 2003; Lladós et al., 2001; Berretti et al., 2004, 2007]. In all these works, the selection of splitting, merging, or both operations is problem-dependant. To the best of our knowledge, there is no formal problem definition for the general case.

In other works, multivalent GM is defined simply as one vertex being matched with a cluster of vertices in another graph [Zaslavskiy et al., 2010]. Whatever how multivalent GM is defined, one common point is that the edge relations induced by vertex splitting and merging are not precisely considered. They concentrate mostly on vertex-to-vertex correspondence. It is shown in Table 2.4.

Another point is that there is no clear distinction between multivalent GM and many-to-many matching in some works [Lyu et al., 2020; Kpodjedo et al., 2013]. In [Lyu et al., 2020], although many-to-many mapping is used, the condition that each operation is applied exactly on each vertex/edge is dropped. So, how to define multivalent matching, many-to-many matching depends a lot on the user's target, and it can lead to more complicated cases. In this thesis, we focus on clarifying the edge relations induced by vertex splitting and merging for one-to-many and many-to-one mappings.

References	Type	Symbolic label	Numeric label	Vertex cost	Edge cost	Edge relation
[Sammoud et al., 2005, 2006]	OTM	✓		✓	✓	✓
[Sorlin and Solnon, 2005; Sorlin et al., 2007]	MTO OTM	✓		✓	✓	✓
[Ambauen et al., 2003]	MTO OTM	✓	✓	✓		
[Berretti et al., 2004, 2007]	MTM	✓	✓	✓	✓	✓
[Cesar et al., 2002]	MTO	✓	✓	✓		
[Cesar Jr et al., 2005]	MTM	✓	✓	✓		
[Deruyver et al., 2005]	MTO	✓	✓	✓		
[Kimelman et al., 2009]	OTM MTO		✓	✓		
[Kpodjedo et al., 2010, 2013, 2014]	MTM	✓	✓	✓		
[Lladós et al., 2001]	OTM MTO	✓	✓	✓		
[Lyu et al., 2020]	MTM	✓	✓	✓		
[Lyu et al., 2019]	MTO	✓	✓	✓		
[Maretic et al., 2020]	OTM		✓	✓		
[Grapa et al., 2019]	MTM			✓		
[Hiransakolwong et al., 2004]	MTM	✓	✓	✓		
[Keselman et al., 2003]	MTM	✓	✓	✓		
[Osmanlıoğlu and Shokoufandeh, 2017]	MTM	✓	✓	✓		
[Zaslavskiy et al., 2010]	MTM	✓	✓	✓		
[Zhang et al., 2005, 2006]	MTM	✓		✓		

Table 2.4: Summary of works that deal with not only univalent matching but also multivalent matching. Type of multivalent matching is specified: One-To-Many (OTM), Many-To-One (MTO), and Many-To-Many (MTM). Other aspects are also mentioned: types of labels, vertex/edge cost for splitting/merging case, and consideration of edge relation.

2.3.2 Examples of multivalent GM problem in different domains

Due to the properties of multivalent GM, it is applied in some specific domains that many-to-one or one-to-many is meaningful, and one-to-one mapping is not robust enough. It may be in PR, biometrics, network analysis. In almost applications, the multivalent GM problem has been employed for performing graph retrieval or classification tasks. Here we provide some popular applications with some examples.

Image analysis and pattern recognition Graphs are usually used to represent the objects or patterns in the images. Vertices are the principal components such as objects (multi-object tracking), parts of the object (semantic graph). Vertex's attributes are often features related to objects/parts such as coordinates, special features. Edges are relations between vertices. Edge's attributes can be the type of relations between vertices. The multivalent GM in these cases is used to express one-to-many or many-to-one relationships between compared objects. In image segmentation, it is the case for under/over-segmented images in post-processing step [Ambauen et al., 2003; Boeres et al., 2004; Deruyver et al., 2005]. In multi-object tracking, it is the case of occlusion where two or more objects are

occluded in consecutive frames [Perera et al., 2006; Henriques et al., 2011; Park et al., 2014; Fan et al., 2016]. In other cases, subpart correspondence is not always one-to-one due to different views, scales, and distortion as in [Cour et al., 2007; Morrison and Zou, 2015]. In these works, the dissimilarity from multivalent GM result will be the criterion to do next tasks such as retrieval, classification [Champin and Solnon, 2003].

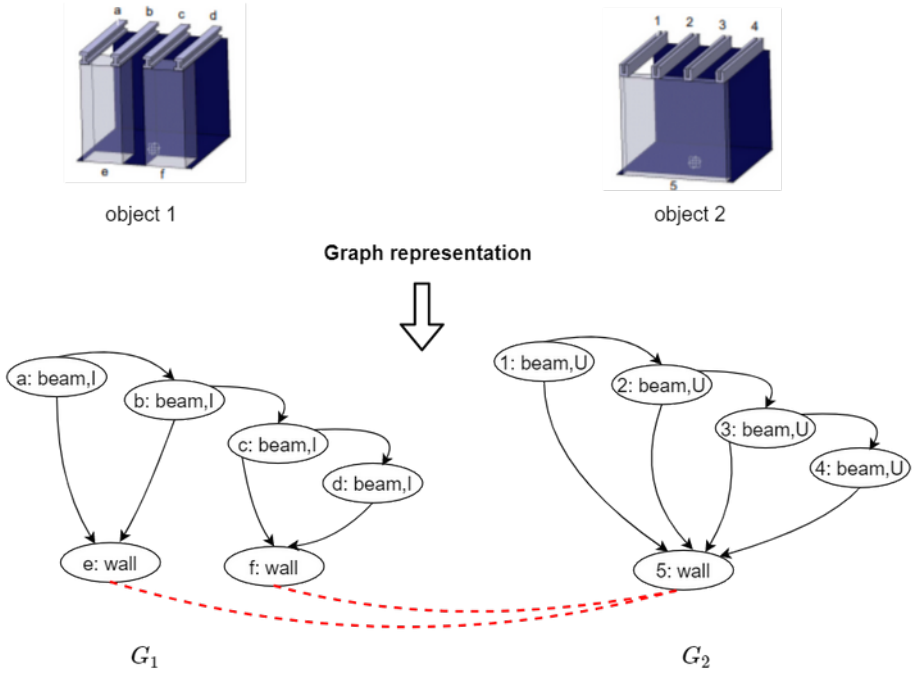


Figure 2.14: Application of multivalent GM in object design where two parts of object 1 equivalent to one part of object 2 (red dashed lines). Figure is taken and modified from one in [Champin and Solnon, 2003].

Handwritten document analysis Graphs usually model segmented words in images. Vertices are key points, strokes, or elements (words) in a sequence. Edges are links between these elements. For this domain, multivalent GM is employed to deal with degraded handwritten and printed document images where outliers or noisy elements can decrease the accuracy of one-to-one mapping. With a given graph query, the multivalent GM can help to give more precise correspondences [Sanfeliu and Fu, 1983; Khurshid et al., 2009; Mondal et al., 2016].

Network analysis In this field, graphs describe networks of people or social networks or traffic congestion. In citation network, it happens that one or more authors have the same names on the papers, or same authors with different display names [Shin et al., 2014]. So multivalent GM is employed to merge or split these relations to facilitate citation step or database management. In mobility service, many-to-one matching is necessary when having traffic congestion [Meshkani and Farooq, 2021] to serve on-demand shared mobility system. In diagram matching for design comparison, multivalent GM is used to retrieve similar

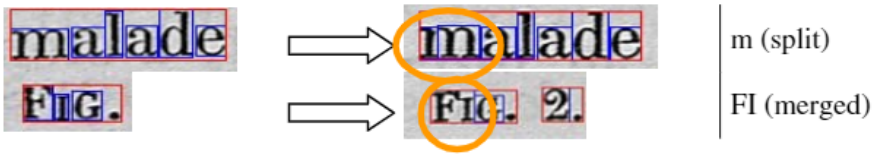


Figure 2.15: Query words and some instances of them which were not spotted without Merge-split in word comparison in [Khurshid et al., 2009].

diagrams or models [Kimelman et al., 2009; Kpodjedo et al., 2013]. Similar applications can be found in [Zhang et al., 2005; Dong et al., 2020].

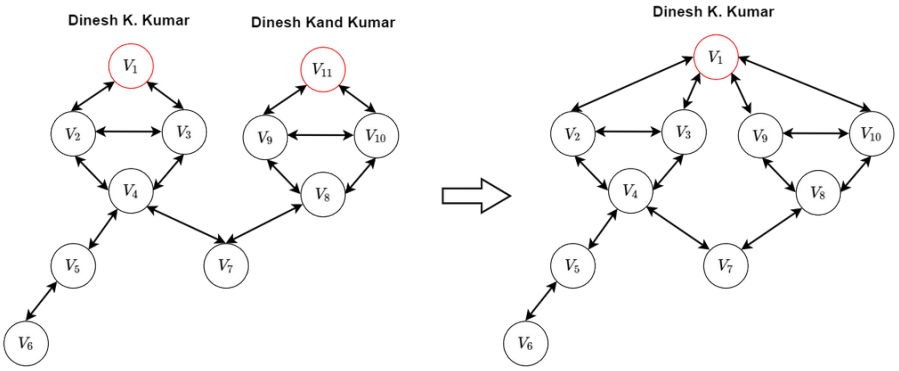


Figure 2.16: Examples of merging the information of vertices denoting the same author in [Shin et al., 2014].

Other applications Beside the above applications, multivalent is also applied in other domains such as biometrics with fingerprint recognition [Shen et al., 2001], operational research with task allocation [Dutta and Asaithambi, 2019], economic with business process comparison [Sebu and Ciocarlie, 2015], and security with individual identification [Du et al., 2010].

2.4 Summary and perspectives

In this chapter, we have presented the interest of another iniquitous kind of data that is known to be more complex than ordinary numerical data, called graphs. We are interested in them because of their power for generalizability and interpretability [Angles and Gutierrez, 2008; Bullmore and Bassett, 2011]. For this data, the graph comparison problem is inevitable to evaluate the difference between objects. A GM algorithm is employed to achieve this purpose. Within the GM family, there are two main categories: exact GM and error-tolerant GM. Their common sub-problems are shown in Table 2.5 and Figure 2.17 in terms of constraints and complexity. In a real-life setting, the second category is more flexible and generic than exact GM due to its capacity to tolerate attributes and topologies.

For that reason, error-tolerant GM is prior in this thesis, but it is more complicated to solve (Figure. 2.17).

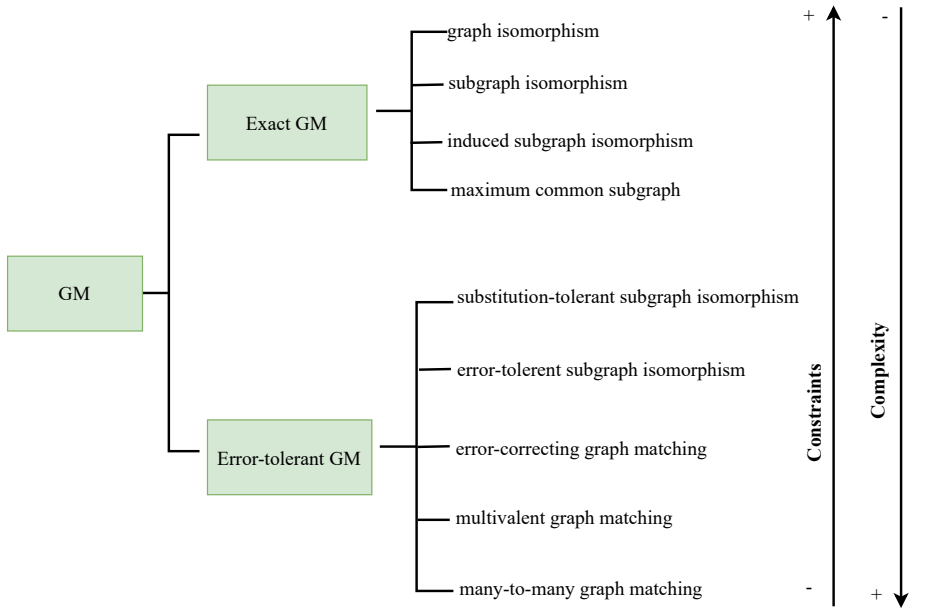


Figure 2.17: Graph matching categories with respect to constraints and complexity [Abu-Aisheh, 2016].

	GM problem	Complexity
Exact GM	Graph isomorphism	$\mathcal{NP}$
	Subgraph isomorphism	$\mathcal{NP}$-complete
	Induced subgraph isomorphism	$\mathcal{NP}$-complete
	Maximum common subgraph	$\mathcal{NP}$-hard
Error-tolerant GM	Substitution-tolerant subgraph isomorphism	$\mathcal{NP}$-hard
	Error-tolerant subgraph isomorphism	$\mathcal{NP}$-hard
	Error-correcting graph matching	$\mathcal{NP}$-hard
	Multivalent graph matching	$\mathcal{NP}$-hard
	Many-to-many graph matching	$\mathcal{NP}$-hard

Table 2.5: Categories of GM problems and their complexities [Darwiche, 2018].

Inside error-tolerant GM, there are several sub-problems aforementioned. In general, most of the sub-problems are univalent matching. That means it allows to map each vertex/edge in one graph to a maximum of one vertex/edge in another graph. Nevertheless, in some specific applications, one-to-one correspondence is not strong enough to cover all complicated scenarios such as over-segmentation/under-segmentation in image processing, occlusion/inclusion in video tracking [Ambauen et al., 2003; Fan et al., 2016]. Therefore, multivalent GM is more appropriate in these cases. It allows many-to-one, one-to-many in addition to one-to-one so that it can deal with more noisy datasets and more realistic

circumstances. Therefore, we will focus on studying the multivalent GM in this thesis.

Besides the advantages discussed, we find some challenges for the multivalent GM problem in our literature review. Firstly, the problem is not well defined as other problems in error-tolerant GM. Generally, most works consider multivalent GM as an extension of GED by adding splitting and merging. But, to the best of our knowledge, none of them has given a formal definition. Secondly, the edge relations prompted by multivalent mapping of vertices have not been well studied, even ignored in several works (Table 2.4). Generally, the edges present the structure of the object. Therefore, studying edge relations induced by vertex mapping plays an important role when doing graph matching. Also, Definition 16 can work with graphs having only symbolic labels. That is not very flexible, for graphs with numerical attributes in particular. Therefore, one needs other manners to define multivalent matching in more elastic ways that can deal with any type of graph.

From these challenges, we would like to propose contributions on the following aspects:

- Provide a more clear and precise definition of multivalent GM by using a GED formulation
- Consider the edge relation in the case of multivalent mapping
- Develop a meta-heuristic approach to solve multivalent GM based on GED

Chapter 3

Optimization techniques for error-tolerant graph matching: from general approaches to meta-heuristic algorithms

Contents

3.1 Synthesis of popular approaches for error-tolerant graph matching 45

3.2 A focus on Ant Colony Optimization 66

3.3 Conclusion on the state-of-the-art 80

Abstract

As discussed previously, error-tolerant GM is more flexible and adaptable in the real context than the exact GM. Therefore, in this chapter, we focus mainly on error-tolerant GM. First of all, we revise the popular techniques for solving the error-tolerant GM. Although neural network for GM is less well-known than other techniques, more and more works try to apply this technique for solving GM problem nowadays. Hence, we also give attention to this method for the GM problem. Then, we present the connection between the GM problem and the assignment problem. We see that the meta-heuristic algorithms are robust to tackle assignment problems. There exist numerous works that employ meta-heuristic methods for the GM problem. Among these methods, we concentrate on ant colony optimization due to its flexibility and efficiency for assignment problem. Therefore, there is a separate section devoted to explain ACO in more detail. Finally, we withdraw some conclusions from the literature review and give our main research directions for the following chapters.

3.1 Synthesis of popular approaches for error-tolerant graph matching

In this section, we focus on reviewing popular techniques for solving the error-tolerant GM problems. These techniques can be clustered into two main categories: exact (optimal) methods and inexact (sub-optimal) methods. The exact methods seek for an optimal solution for the GM problem. Meanwhile, the inexact methods try to obtain approximate solutions for the GM problem [Conte et al., 2004]. Because GM problem is already $\mathcal{NP}-$hard, the exact methods usually request an exponential running time as the graph sizes increase [Darwiche, 2018]. As consequence, the exact methods can only work on small graph sizes. For example, the A* algorithm designed by Hart et al. [Hart et al., 1968] will be intractable with large graph size [Darwiche, 2018]. In the previous chapter, we declared our interest on the multivalent GM problem, and its complexity is even higher than other types of GM problems (Fig. 2.17). As a result, the exact methods may not be our favor for this problem. Hence, we intend to explore more the suboptimal methods to deal with the multivalent GM problem rather than the exact ones. Although there are several approximate approaches that tackle the error-tolerant GM problems [Conte et al., 2004; Yan et al., 2016; Laura et al., 2021], we concentrate intentionally on works that relate to our main problem-multivalent GM.

The subsections will follow the taxonomy of GM methods as presented in Laura et al. [Laura et al., 2021]. For each group in the taxonomy, we present the primary principles to give resolution to the error-tolerant GM problems, the related works, and the general pipeline for each category. Moreover, the advantages as well as limitations for each group will be discussed. Last but not least, we pay more attention to the methods dealing with multivalent mapping. Note that works concerning many-to-many matching can imply multivalent GM inside. Therefore, some works linked to many-to-many matching are also mentioned in this section. From now on, the term 'vertex' refers to an element of a graph, while the term 'node' is for an element of a search tree for the sake of clarity.

3.1.1 Tree search-based approaches

Tree search is known to be one of the foremost approaches for solving a GM problem [Tsai and Fu, 1979; Shapiro and Haralick, 1981]. The tree-based approaches proceed by enumerating implicitly all possible solutions, but they do not explicitly evaluate all of them by means of an ordered tree [Raveaux, 2019]. An ordered tree is a rooted tree in which the children of a node are ordered specifically [Korte et al., 2011]. Then, the ordered tree is built dynamically at run time. It starts with a root node where matching is empty with zero cost. Then, the child nodes are constructed iteratively at run time each child node p is a partial matching. A leaf of the tree presents a feasible matching. At each iteration, the decision of the next child node to be expanded depends on the predefined strategies. They are:

- Depth-first: the most promising node among child nodes of p is selected,
- Breadth-first: the most promising node that is at the same level of p is selected,

- Best-first: the most promising node is selected without constraint (at the same level or not).

To evaluate whether a child node p is promising or not, an evaluation function f is requested. Regularly, $f = g(p) + h(p)$, where $g(p)$ is the cost of the partial matching accumulated so far, $h(p)$ is a heuristic that estimates the cost from p to a leaf node. Therefore, function f presents an estimated cost for a complete solution, and is referred as a lower bound $lb(p)$ [Fischer et al., 2017]. For exact approaches, if $h(p)$ is admissible, i.e., the estimation of future cost $h(p)$ is lower than, or equal to, the real cost, it is guaranteed to obtain an optimal matching [Raveaux, 2019; Riesen et al., 2007a; Abu-Aisheh et al., 2015]. For inexact or heuristic approaches, the tree search can be truncated by limiting the number of tree nodes p in memory [Neuhaus et al., 2006], or by limiting the time to generate the solutions [Abu-Aisheh et al., 2016]. Figure 3.1 presents an overall pipeline for tree-based approaches for the exact and inexact approaches. For exact approaches, the matching at the end is optimal. For inexact approaches, there are more constraints to prune the search tree, and the produced matching is usually sub-optimal. For GED, the matching is equivalent to the edit path λ.

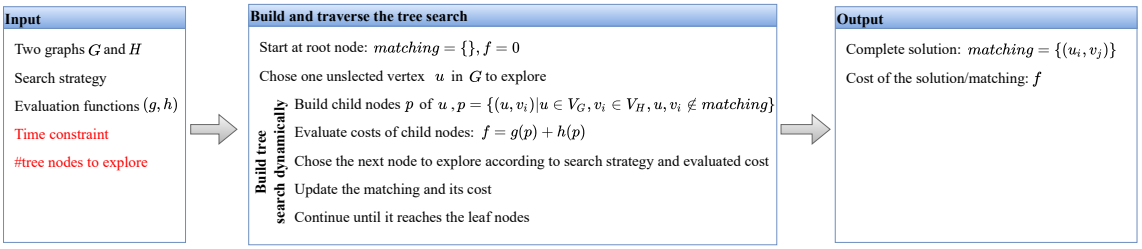


Figure 3.1: A general pipeline of tree-based approaches for solving GM problem, G and H are two labeled graphs predefined, inputs in red color are for heuristic tree-based approaches.

An illustrative example is given in Fig. 3.2 taken from [Riesen, 2015]. Let g_1 and g_2 be two compared graphs, the vertex labels are integers and there is no label on edges. We want to compute the GED between g_1 and g_2. The cost of the edit path $g(\lambda)$ (partial or complete) equals the sum of the costs derived from vertex edit operations and the cost of all edge operations that can be induced by vertex operations carried out so far. To be simple, the heuristic function $h(p)$ is set to 0, i.e., there is no heuristic information employed in the example. The accumulated costs are presented in the tree search vertices. The cost functions for edit operations are as follows.

- $c(u \rightarrow v) = |l_u - l_v|$
- $c(u \rightarrow \varepsilon) = c(\varepsilon \rightarrow v) = 1$
- $c((u_i, u_j) \rightarrow (v_k, v_l)) = 0$
- $c((u_i, u_j) \rightarrow \varepsilon) = c(\varepsilon \rightarrow (v_k, v_l)) = 1$

The tree starts at level 0-root node with $\lambda = \{\}$ and $g(\lambda) = 0$. At level 1, it can start with the vertex u_1 in g_1, and creating child nodes $p = \{(u_1, v_k), \forall v_k \in g_2\} \cup (u, \varepsilon)$. For instance, at node (u_1, v_2), the cost is 1 because of $c(u_1 \rightarrow v_2) = |1 - 2| = 1$. Calculation for other nodes at the same level is similar. Suppose that we use depth-first search in this example. At level 1 (u_1), there are three nodes with equal costs. For instance, if (u_1, v_2) is selected, the current edit path is $\lambda = \{u_1 \rightarrow v_2\}$. Then other child nodes are built from the selected nodes, without vertices already in the current edit path, i.e, u_1 and v_2. So, at level 2, all possible combinations with u_2 is v_1, v_2, ε. Then, the evaluation process is performed. For instance, at child node $p = (u_2, v_1)$, we have these following costs: vertex substitution $c(u_2 \rightarrow v_1) = |2 - 3| = 1$, edge substitution induced by two substitutions of vertices: $c((u_1, u_2) \rightarrow (v_1, v_2)) = 0$. Therefore, with $\lambda = \{u_1 \rightarrow v_2, u_2 \rightarrow v_1\}$, the accumulated cost is $1 + 1 + 0 = 2$. For the next level, the operation $u_3 \rightarrow v_3$ costs 3 due to one vertex substitution ($c(u_3 \rightarrow v_3) = 1$), one edge deletion ($c((u_2, v_3) \rightarrow \varepsilon) = 1$), and one edge insertion ($c(\varepsilon \rightarrow (v_2, v_3)) = 1$). Hence, the accumulated cost of $\lambda = \{u_1 \rightarrow v_2, u_2 \rightarrow v_1, u_3 \rightarrow v_3\}$ so far is 5. Finally, by adding $(u_4 \rightarrow \varepsilon)$ to the edit path, it costs 3 with one vertex deletion and two induced edge deletions. Therefore, the final edit path has a cost of 8. In this example, the optimal edit path is displayed in bold arrows, i.e,

$$\lambda_{min} = \{u_1 \rightarrow \varepsilon, u_2 \rightarrow v_3, u_3 \rightarrow v_2, u_4 \rightarrow v_1\}$$

with a minimum cost of 4.

For multivalent GM, the tree search-based methods are also utilized in some works. In [Lladós et al., 2001], Llados et al. proposed to use a brand-and-bound algorithm to search for the solution of the matching problem. Three edit operations, substitution, shift, and merge, are utilized for the error-tolerant subgraph matching problem between Region Adjacency Graphs (RAGs). The shift operation is done in the input graph and it tries to combine two regions sharing the same subtrings. Thus, this operation is used to preserve the structural relationships between regions. The RAGs are designed to reduce the size of the normal graphs in handwritten drawn images. These images include gaps or broken lines due to noise and inaccurate stokes. Accordingly, the authors introduce a particular type of edge, called bridges, to define candidates for merging. The proposed method solves graphs with a few hundred vertices in the input graph and around ten vertices in the model graph. However, the cost computation of vertex merging and the edge relations implied are not explicit.

In [Ambauen et al., 2003], Ambauen et al. introduced an exact tree-based method for identifying diatoms automatically. Each diatom is represented as a labeled and directed graph. The approach is based on GED edit operations with splitting and merging operations for vertices. By introducing two additional edit operations, the authors can do the multivalent matching to match the original images with the over-segmented images. Nevertheless, it can only solve small graphs due to the complexity of the algorithm [Sorlin et al., 2007]. Unfortunately, the induced edge relations from vertex splitting and merging are not discussed. The costs for vertex splitting and merging are 0, which implies a priority for these operations even higher than the vertex substitution.

In [Hiransakolwong et al., 2004], Hiransakolwong et al. presented a tree search-based approach for shape recognition. The shapes are transformed in skeletal graphs. They are undirected and labeled on vertices and edges. Many-to-many matching (multivalent

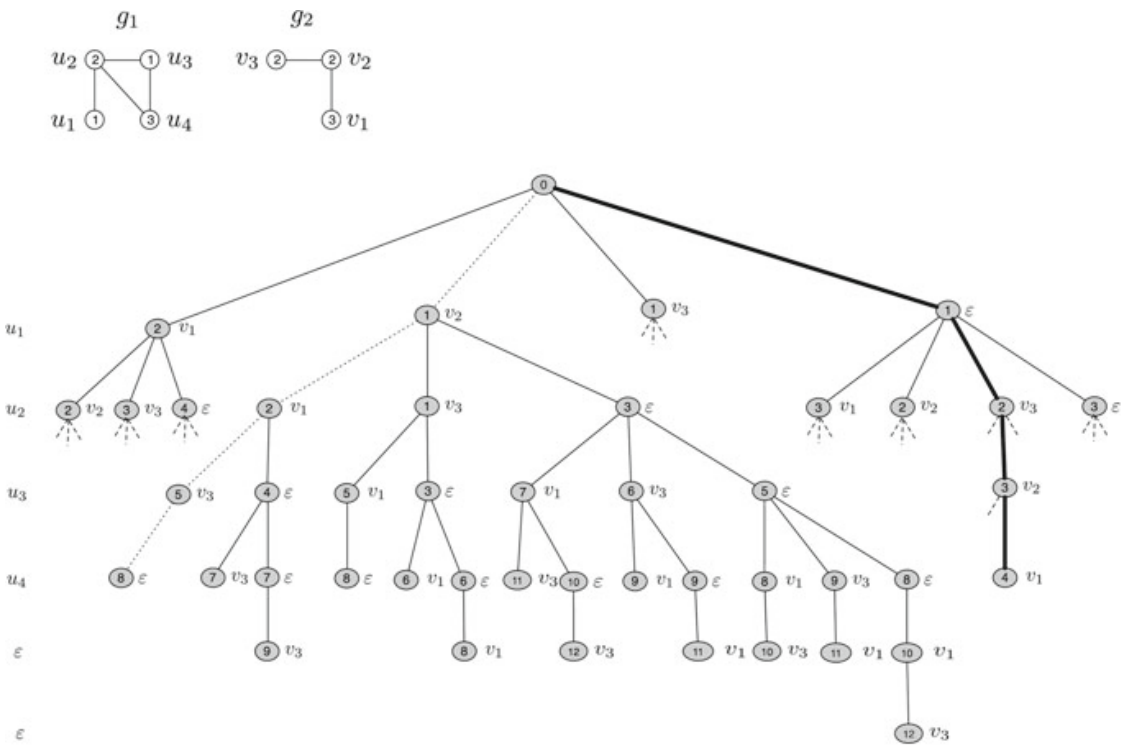


Figure 3.2: Part of the search tree for GED computation between graphs $g1$ and $g2$, image is taken from [Riesen, 2015]. Bold arrows indicates the optimal edit path in this case with a minimum cost of 4.

matching included) is proposed to tackle occluded and included images. The depth-first search strategy is used to traverse the search tree. Although some results are more robust to ones of the directed acyclic graph method, the matching method concentrates principally on the vertex level, not the edge level. Hence, the information at edge level may be omitted.

3.1.2 Continuous relaxation-based approaches

In the previous section 3.1.1, the tree search-based approaches are proposed to solve the GM problem directly in the discrete domain. Nevertheless, there are other attempts to tackle the GM problem indirectly in the continuous domain. They are considered as continuous relaxation-based approaches [Conte et al., 2004; Laura et al., 2021]. Specifically, it transforms a discrete optimization problem into a continuous one by relaxing discrete mapping constraints. Then, a heuristic solver is employed to seek the continuous solution. After, the found solution will be projected to the initial discrete problem by some approximations. Although the algorithms used to find solutions for the relaxed continuous problems do not guarantee an optimal solution, they are rather time-saving [Conte et al., 2004; Zhou and De la Torre, 2015]. In particular, it can reduce the computational cost polynomially respecting the graph size. However, for this method, the estimation quality is sensitive to the solution initialization. Also, the step to convert the continuous solution

to discrete one can introduce an additional level of approximation [Conte et al., 2004]. According to Ma et al., these approaches can be divided into three main categories in the field of image matching: convex relaxation, convex-concave relaxation, and continuous relaxation [Ma et al., 2021]. A general framework for these main steps is illustrated in Fig. 3.3. Subsequently, we detail more how to solve GM in the continuous domain with various relaxation techniques.

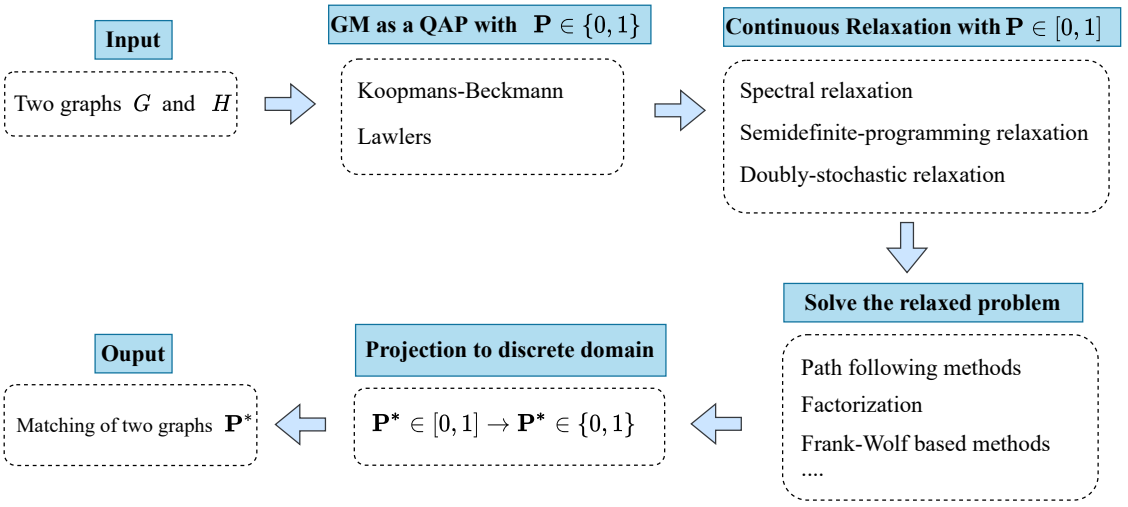


Figure 3.3: General framework of solving GM with continuous relaxation approach, where $\mathbf{P}$ is the permutation matrix of two input graphs.

Quadratic Assignment Problems (QAP) for GM Roughly speaking, the GM problem seeks an alignment between vertices in G and H, which is optimal in some sense [Zhang et al., 2019]. The GM problem is known to be equivalent to the $\mathcal{NP}-$hard QAP [Raveaux, 2019]. It is important to formulate the GM problem as a QAP [Loiola et al., 2007] to solve the GM problem with continuous optimization. Despite existing different forms of QAP, two formulations are preferred: one is of Lawler [Lawler, 1963], and another is of Koopmans-Beckmann [Loiola et al., 2007]. To transform into these QAPs, two graphs must have the same orders, i.e., same number of vertices. If not, the dummy vertices are added to satisfy the assumption. The Koopmans-Beckmann's QAP aims to maximize the Frobenius inner product between two adjacency matrices after permutations [Koopmans and Beckmann, 1957]. The Frobenius inner product is component-wise inner product of two matrices and returns a number [Jans, 1959]. The Lawler's QAP seeks the similarity maximization between the vertex attribute sets and the edge attribute sets [Lawler, 1963]. In Zhou and De la Torre (2015), they investigated the relation of two QAP formulations [Zhou and De la Torre, 2015]. It is proved that Koopmans-Beckmann's QAP is a special case of Lawler's. Nevertheless, one problem of Lawler's QAP is that the matrix $\mathbf{K}$ is quadruple in respect of n. $\mathbf{K}$ is a square symmetric positive matrix such that $\mathbf{K}_{ik,jl}$ measures how well every pair $e_{ij} \in E_G$ matches with $e_{k,l} \in E_H$. For pair that do not form edges, their corresponding entries are set to 0. The diagonal entries represent vertex-to-vertex score. The significant increase of $\mathbf{K}$ regarding n will be a challenge to deal with

large graphs [Zhang et al., 2019].

Different relaxations on the constraints for GM From the framework in Fig. 3.3, there exist three main groups to do relaxation for the GM problem. They are spectral relaxation, Semi-Definite Programming (SDP), and doubly-stochastic relaxation [Zhou and De la Torre, 2015].

The first group includes methods that approximate the permutation matrix $\mathbf{P}$ with an orthogonal one, i.e., $\mathbf{P}^T\mathbf{P} = \mathbf{I}$. Thanks to this constraint, the GM now can be solved as an *eigen-value problem* [Umeyama, 1988; Scott and Longuet-Higgins, 1991; Caelli and Kosinov, 2004; Fan et al., 2020]. Nonetheless, these methods are only applicable for Koopmans-Beckmann's QAP, a special case of QAP. In [Leordeanu and Hebert, 2005], Leordeanu and Hebert propose to relax P to be of unit length, i.e., $\|\mathbf{P}\|_2 = 1$. Then, $\mathbf{P}$ can be efficiently computed as a leading eigen-vector of $\mathbf{K}$. Consequently, it is more general and can deal with more complex problem in computer vision. Later, Cour et al. impose an additional affine constraint to solve more general spectral problem. The numerical results show that it obtains better approximation to the original problem [Cour et al., 2007].

SDP is also used for approximating GM. Generally, the objective of GM is reformulated by *introducing a new variable* $\mathbf{Y} \in \mathcal{R}^{n^2 \times n^2}$ subject to $\mathbf{Y} = \text{vec}(\mathbf{P})\text{vec}(\mathbf{P})^T$ [Torr, 2003; Schellewald and Schnörr, 2005]. After, SDP approximates GM by relaxing the non-convex constraint on $\mathbf{Y}$ as a convex semi-definite one, i.e., $\mathbf{Y} - \text{vec}(\mathbf{P})\text{vec}(\mathbf{P})^T \geq 0$. When $\mathbf{Y}$ is calculated, $\mathbf{P}$ can be estimated by a randomized algorithm [Torr, 2003] or a winner-take-all strategy [Schellewald and Schnörr, 2005]. According to [Torr, 2003], although SDP shows a robust speed up theoretically [Goemans and Williamson, 1995], it is too costly in practice. That is because $\mathbf{Y}$ increases doubly as the problem size increases. To overcome this problem, Bravo Ferreira et al. proposed extra relaxations [Bravo Ferreira et al., 2018]. Specifically, the positive semi-definiteness of $\mathbf{Y}$ is scarified. Instead of that, they enforce only positive semi-definiteness of sub-matrices in $\mathbf{Y}$. Moreover, the cliques of the graph are considered rather than the edges. However, one need to define a maximal clique size beforehand to avoid spending time to list all cliques. The acceleration is achieved if one of the compared graph is sparse.

The last group contains a wide range of methods that relaxes $\mathbf{P} \in \mathcal{D}$ to be *a doubly stochastic matrix*, where $\mathcal{D}$ is the convex hull of the permutation matrix [Zhou and De la Torre, 2015], i.e., $\mathcal{D} = \{\mathbf{P}|\mathbf{P} \in [0,1], \mathbf{P}\mathbf{1}_{n_H} \leq \mathbf{1}_{n_G}, \mathbf{P}^T\mathbf{1}_{n_G} = \mathbf{1}_{n_H}\}$ with n_G, n_H being the number of vertices in G and H, respectively. With this constraint, optimizing GM can be considered as a non-convex quadratic assignment problem. Consequently, there exists numerous strategies to seek an approximate solution [Zhou and De la Torre, 2015; Raveaux, 2019]. Path following method is one of the popular strategies. In [Zaslavskiy et al., 2008; Liu and Qiao, 2013; Bougleux et al., 2017], the Graduated NonConvexity and Concativity Procedure (GNCCP) which is a path following algorithm is used to approximate the solution of QAP. It considers a convex-concave relaxation through the modified quadratic function [Liu and Qiao, 2013]:

$$F_\zeta(\mathbf{P}) = (1 - |\zeta|)F(\mathbf{P}) + \zeta(\mathbf{P}^T\mathbf{P})$$

where $\zeta \in [-1, 1]$. If $\zeta = 1, F_\zeta(\mathbf{P}) = \mathbf{P}^T\mathbf{P}$ is fully convex. If $\zeta = -1, F_\zeta(\mathbf{P}) = -\mathbf{P}^T\mathbf{P}$ is fully concave. GNCCP starts with $\zeta = -1$ and it leads to maximization of a concave

problem. For GNCCP, it can begin with any initial solution without affecting its result [Raveaux, 2019]. At each iteration, GNCCP interpolates smoothly concave and convex relaxations from -1 to 1 with step size α. Regarding ζ at each iteration, the maximization of F_ζ is obtained by the Integer Projected Fixed Point (IPFP) [Leordeanu et al., 2009].

Multivalent GM related Most of works mentioned previously are for univalent GM where one vertex can be associated with at most one vertex. Nonetheless, there are few works that propose continuous relaxation for many-to-many GM. It means that clusters of vertices in one graph can match to clusters of vertices in other graph [Zaslavskiy et al., 2010]. The matching can also imply one-to-many and many-to-one. One representative work for many-to-many GM using continuous relaxation is of [Zaslavskiy et al., 2010]. The one-to-one classical GM problem can be formulated as a discrete optimization problem below:

$$\min_{\mathbf{P}} \parallel \mathbf{A_G} - \mathbf{P}\mathbf{A_H}\mathbf{P}^T \parallel_F^2 \text{ s.t } \mathbf{P} \in \mathcal{P}_{oto} \text{ with} \tag{3.1}$$
$$\mathcal{P}_{oto} = \{\mathbf{P} \in \{0,1\}^{n \times n}, \mathbf{P}\mathbf{1}_n = \mathbf{1}_n, \mathbf{P}^T\mathbf{1}_n = \mathbf{1}_n\}$$

where $\mathcal{P}_{oto}$ is set of permutation matrices for one-to-one matching.

The problem of finding one-to-many (many-to-one) matching can be formulated with the same criterion as in 3.1 but with the modified constrains:

$$\mathcal{P}_{otm}(n_G, n_H) = \{\mathbf{P} \in \{0,1\}^{n_G \times n_H}, \mathbf{P}\mathbf{1}_{n_H} \leq k_{\max}\mathbf{1}_{n_G}, \mathbf{P}^T\mathbf{1}_{n_G} = \mathbf{1}_{n_H}\},$$

with the assumption that G contains more vertices than H, $k_{\max}$ is the maximum number of vertices that can be associated to a single vertex. With the above constraint, the sum of each row of $\mathbf{P}$ can be greater than 1, and the non-zero elements of the i-th row of $\mathbf{P}$ denote vertices of H being matched to the i-th vertex of G. Then a virtual graph S with n_S vertices is built to do many-to-one matching from G to S, and many-to-one matching from H to S. Therefore, the authors express many-to-many matching as a double many-to-one matching as in Fig. 3.4.

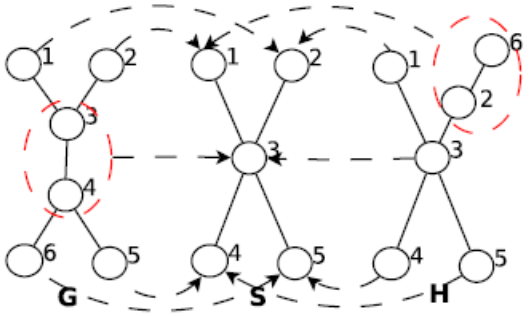


Figure 3.4: Doing many-to-many matching from G to H as a double many-to-one matching of both graphs to the virtual graph S. Image is taken from [Zaslavskiy et al., 2010].

From the virtual graph S, they define

- $\mathbf{P}_1 \in \mathcal{P}_{mto}(n_G, n_S)$ as many-to-one matching $G \to S$
- $\mathbf{P}_2 \in \mathcal{P}_{mto}(n_H, n_S)$ as many-to-one matching $H \to S$.

Then, the many-to-many GM is considered as an optimization problem where we need to find $S, \mathbf{P}_1$ and $\mathbf{P}_2$ such that the difference between S and $\mathbf{P}_1\mathbf{A_G}\mathbf{P}_1^T$, and between S and $\mathbf{P}_1\mathbf{A_H}\mathbf{P}_2^T$ are minimized. In other word, one needs to minimize the following objective function:

$$F(\mathbf{P}_1, \mathbf{P}_2) = \| \mathbf{P}_1\mathbf{A_G}\mathbf{P}_1^T - \mathbf{P}_1\mathbf{A_H}\mathbf{P}_2^T \|_F^2 \tag{3.2}$$

where $\mathbf{P}_1$ and $\mathbf{P}_2$ are defined previously. From 3.2, we seek combinations of merges and permutations which maximize the similarity between G and H. The final problem formulation for many-to-many matching is:

$$\begin{aligned} &\min_{\mathbf{P}_1,\mathbf{P}_2} \| \mathbf{P}_1\mathbf{A_G}\mathbf{P}_1^T - \mathbf{P}_2\mathbf{A_H}\mathbf{P}_2^T \|_F^2 \text{ s.t} \\ &\mathbf{P}_1 \in \{0,1\}^{n_K \times n_G}, \mathbf{P}_1\mathbf{1}_{n_G} \leq k_{\max}\mathbf{1}_{n_K}, \mathbf{P}_1^T\mathbf{1}_{n_K} = \mathbf{1}_{n_G}\}, \\ &\mathbf{P}_2 \in \{0,1\}^{n_K \times n_H}, \mathbf{P}_2\mathbf{1}_{n_H} \leq k_{\max}\mathbf{1}_{n_K}, \mathbf{P}_2^T\mathbf{1}_{n_K} = \mathbf{1}_{n_H}\}, \end{aligned} \tag{3.3}$$

where $n_K = \min(n_G, n_H)$ represents the maximal number of matching clusters. The formulation 3.3 is applicable for directed graphs with weighted edges, self-loops, and labels on vertices.

After, the binary constraints of $\mathbf{P}_1 \in \{0,1\}^{n_K \times n_G}$ and $\mathbf{P}_2 \in \{0,1\}^{n_K \times n_H}$ are relaxed to continuous constraints, i.e., $\mathbf{P}_1 \in [0,1]^{n_K \times n_G}$, and $\mathbf{P}_2 \in [0,1]^{n_K \times n_H}$. The objective function 3.3 is quadratic so that they use conditional gradient (a.k.a. Frank-Wolfe method [Frank and Wolfe, 1956]) to approximate the solution. The Frank-Wolfe algorithm is an iterative first-order optimization algorithm for constrained convex optimization. At each iteration, it minimizes a linear function to approximate the objective function. Its working principle is similar to gradient descent method, but a linear program is solved iteratively [Raveaux, 2019]. The proposed method requires an initialization of $\mathbf{P}_1$ and $\mathbf{P}_2$. Hence, the result can be affected by the initial solution. Also, the discretization step which is done by a greedy scheme of incremental projection can have an impact on the final result. Some promising results are obtained for letter recognition, deformable object matching, and identification of object composite parts. Nevertheless, the approach is proposed only for weighted graphs, not for graphs with symbolic labels. Moreover, the sensitivity of solution quality to the initialization phase is also a concern [Grapa et al., 2019].

Besides research of Zaslavskiy for many-to-many matching [Zaslavskiy et al., 2010], other works also tackle multivalent GM based on continuous relaxation such as [Maretic et al., 2020; Grapa et al., 2019]. Both works relax the problem into the continuous domain and use the Optimal Transport (OT) to solve the transformed problems. OT bases on the theory of computing distances between probability distributions [Raveaux, 2019]. These distances can be the empirical estimations of the distributions, and they also exploit the geometry of the underlying metric space [Raveaux, 2019; Grapa et al., 2019]. OT is similar to GM because it considers also the interactions (edges) between data. The Gromov-Wasserstein distance is an optimal transport distance and it can be applied to model a graph matching problem. Given two graphs G and H and their respective adjacency

matrices $\mathbf{A_G}, \mathbf{A_H}$, they define $Pm_G \in \mathcal{R}^{n_G}$ be the empirical distributions of vertices in G. So, Pm_G counts the appearance of each vertex in E_G. Pm_H is defined similarly. Then,

$$d(Pm_G, Pm_H) = \min_{Pr \in \Pi(Pm_G, Pm_H)} \sum_{(i,k) \in V_G \times V_H} \sum_{(j,l) \in V_G \times V_H} \parallel A_G^{i,j} - A_H^{k,l} \parallel .Pr(i,j).Pr(k,l)$$

where $Pr \in \mathcal{R}^{n_G \times n_H}$ is the joint probability of vertices $i \in V_G$ and $k \in V_H$. Pm_G is the marginal of Pr such that $Pm_G^i = \sum_{k \in V_2} Pr(i,k)$. $\Pi(Pm_G, Pm_H)$ is the set of all possible joint probabilities. The minimized Gromov-Wasserstein can be obtained by selecting the largest $Pr(i,k)$. Nevertheless, the binary mapping constraints may not be respected [Raveaux, 2019]. As shown in [Grapa et al., 2019], optimal transport is less robust than the proposed method in [Zaslavskiy et al., 2010] in terms of solution quality, yet more time-saving. In [Maretic et al., 2020], graphs are undirected, weighted. In [Grapa et al., 2019], graphs are undirected and unlabeled. Consequently, the proposed methods in these works are not applicable for graphs with symbolic labels.

3.1.3 Meta-heuristic based-approaches

Besides the previous optimization techniques for solving GM problems, meta-heuristic methods are also good candidates. These methods are based on stochastic optimization. Randomness plays a role in the solution-building process to balance exploration and exploitation schemes. Therefore, the solution quality depends on the set of random variables generated [Bianchi et al., 2009]. As a result, it will not guarantee an optimal solution. According to [Blum and Roli, 2003], meta-heuristic can offer good solutions with less computational cost than other optimization algorithms, iterative methods, or simple heuristics. This approach is known to be a good compromise between discrete and continuous optimization techniques. It will be more efficient than tree search-based techniques and not very sensitive to local solutions as continuous optimization methods [Laura et al., 2021]. Nevertheless, due to the nature of randomness, the achieved solutions are not stable at each run time, i.e., non-deterministic [Raveaux, 2019]. Therefore, the approaches are usually run several times to study the stability of the solution.

The meta-heuristic methods can be divided based on their search capacities used: local search and global search. The first group looks for the solution locally, resulting in the convergence to a local minimum. Meanwhile, the second group searches for solutions globally. It has some schemes to escape from the local minima, the randomness strategy particularly [Dorigo and Stützle, 2019]. One example of local search is hill-climbing. One example of global search is Ant Colony Optimization (ACO). Nevertheless, multiple restarting of hill-climbing can help the algorithm to have the global search capacity. Besides, the meta-heuristic algorithms can be categorized according to other criteria as shown in [Yang, 2011].

Fig. 3.5 presents a general framework of using meta-heuristic methods for solving the GM problem. The input is always the two compared graphs G and H. In addition, there are meta-parameters and a fitness function. The meta-parameters are usually the parameters linked to a specific meta-heuristic method. For example, Genetic Algorithm (GA) can have some parameters like the population, i.e., number of chromosomes (*pop*), crossover rate (CR), mutation rate (MR), number of iterations $it_{\max}$ [Kim et al., 2016; Riesen et al.,

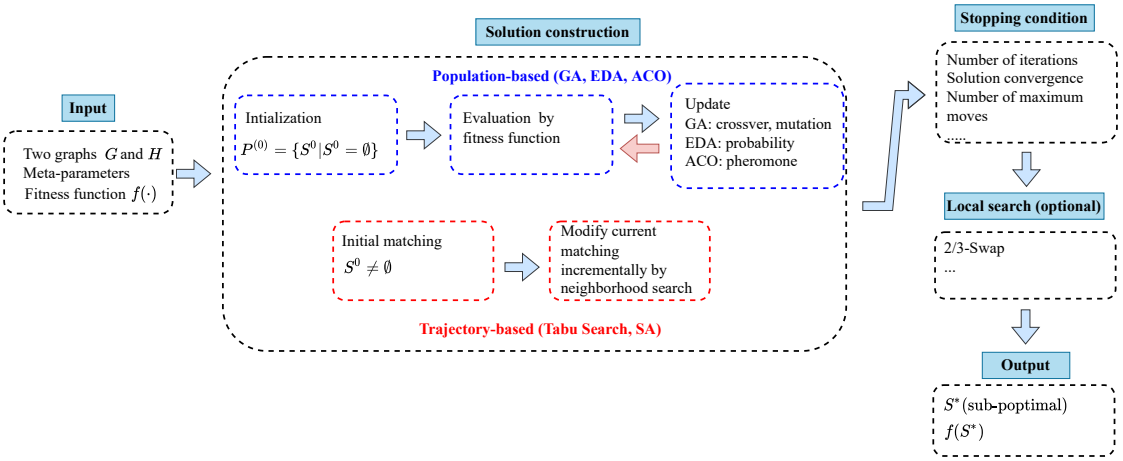


Figure 3.5: A general pipeline of meta-heuristic methods for solving the GM problem.

2014]. Or Tabu search can have the maximum number of allowed moves ($maxMoves$) in [Sammoud et al., 2006]. The meta-heuristic methods can be the main solvers or post-processing methods. For the first case, the method starts an empty matching $S^0 = \emptyset$. Then, it adds a new couple of vertices incrementally based on the fitness function until the matching is complete [Sammoud et al., 2005; Cesar Jr et al., 2005; Kim et al., 2016]. For the second case, the method begins with an initial matching. Then, it modifies the current matching by adding or removing couples of vertices to optimize the fitness function [Riesen et al., 2014, 2017; Kpodjedo et al., 2010, 2013]. For population-based methods, such as GA, ACO, or Estimation of Distribution Algorithm (EDA), they select and update the population (set of matchings) mostly following the principal 'survival of the fittest' [Riesen et al., 2014; Raveaux, 2019]. For trajectory-based methods, such as Tabu Search (TS), Simulated Annealing (SA), they try to decrease the fitness function (in minimizing case) at each iteration to improve the current matching. These steps are repeated until reaching the stopping conditions. They are usually the maximum number of iterations ($it_{\max}$), or the current matching is not able to improve through N-predefined iterations [Sammoud et al., 2006; Riesen et al., 2017]. Then, the Local Search (LS) can be utilized to enhance the found matching. Finally, the method returns the best-found solution along with its cost. For meta-heuristic methods, the solutions are not deterministic. Therefore, the users may run several times to evaluate the method and get a more confident solution [Riesen et al., 2014; Sammoud et al., 2006]. Some popular meta-heuristic methods used for tackling the GM problem will be presented below.

Evolutionary Algorithm (EA)/Genetic Algorithm (GA) Considered as a population-based global search method in the family of meta-heuristic methods, GA has been proposed to solve the error-tolerant GM in several publications [Wang and Ishii, 1998; Fröhlich et al., 2001; Riesen et al., 2014; Kim et al., 2016]. The principle of this approach is to formalize the matchings as chromosomes with fitness values given by the fitness function predefined. In the beginning, it initializes a population including different chromosomes. Then, the population evolves iteratively into other generations of chromosomes [Riesen et al., 2014].

To serve this purpose, several genetic operations, such as selection scheme, crossover, and mutation, are applied to the current generation. These operators ensure that the best chromosomes (matchings) can remain afterward, making the opportunity to diversify the search space. The parameters of GA are $pop, CR, MR, it_{\max}$. The algorithm stops when no more evolution is recorded in the population. That means the values of the fitness function are similar after some generations.

In [Riesen et al., 2014], Riesen et al. used GA to enhance the matching given by the bipartite matching with the Hungarian algorithm. The chromosome presents the vertex assignments of two graphs following the operations of GED. GA builds the initial population from random variations of the initial matching while respecting the constraints of GED. The next generation is derived by retaining E best chromosomes from the previous population and creating $|pop - E|$ new ones. The proposed GA is applied for undirected labeled graphs (both numeric and symbolic). The experiment shows a significant improvement in solution quality compared to the initial one. The computational time is increased but still far below the exact GED method-A^*.

Another work of Kim et al. used GA for error-correcting subgraph isomorphism problems in measuring source codes [Kim et al., 2016]. The graphs are directed and have labels on vertices, including no isolated vertex and no self-loop. In this case, GA constructs the solution incrementally and integrates a local search to enhance the performance. In these works, the considered GM problem is univalent, i.e., one vertex is associated with at most one vertex.

In [Cesar Jr et al., 2005], GA is used for solving multivalent GM of directed labeled graphs. In particular, the authors focus on many-to-one matching to map the over-segmented images to its models in facial feature recognition. Chromosomes in the population are also the vertex assignments between two graphs. And there is no initial matching to start. The initial population is created randomly based on a uniform distribution of possible vertex mappings. The results indicate an inferior GA's performance compared with EDA in terms of solution quality. Concerning multivalent matching, the cost of edge operations induced by vertex merging is computed solely on the graph of the over-segmented images, not on the model graphs. Furthermore, the cost of vertex merging is not explicit.

Tabu search (TS) is a trajectory-based local search algorithm in the meta-heuristic family [Yang, 2011]. Beginning with an initial solution (matching) in the search space, a TS moves iteratively from the current matching to a neighbor one. At every iteration, TS selects the best neighbor concerning the cost given by the fitness function. Meanwhile, it tries to avoid returning towards matching that is visited earlier. Therefore, a tabu list is built to manage couples of vertices already inserted into the matching (solution) [Cesar Jr et al., 2005]. That means the list has a length k to memorize the last k moves. The length of this list can be fixed or dynamic. In [Cesar Jr et al., 2005], it is fixed, yet it is flexible in [Sammoud et al., 2006] called reactive TS. Kpodjedo et al. proposed TS to enhance the solution given by the greedy approach for approximate GM of undirected graphs [Kpodjedo et al., 2010]. It tries to maximize the score function whose terms consider both vertex matching and edge matching. Promising results are obtained, but it shows

that TS has a very poor performance without the initial greedy matching. Moreover, the synthetic graphs in the experiments are relatively similar. And the results show that the score function is not very good to capture the structural variations. The univalent GM is studied in this work. Following that, the authors develop TS for many-to-many GM for directed and labeled graphs in the context of diagram comparison. Nevertheless, only the symbolic label is considered. The edge relations induced by vertex merging are not explained explicitly. Likewise the previous work, TS starts with an initial partial matching called perfect matches and tries to complete the matching.

Simulated Annealing (SA) is another trajectory-based search meta-heuristic [Yang, 2011]. Its working principle is inspired by the metal annealing process in material science [Kirkpatrick et al., 1983]. SA has been proposed to solve difficult optimization problems [Kirkpatrick et al., 1983]. The basic idea of SA for tackling optimization problems is beginning with a (random) initial solution and then randomly disturbing it [Riesen et al., 2017]. If the resulting solution is better than the previous one, it will replace the previous one and be used for the next step. Nevertheless, the worse solution can be accepted with a certain probability. That depends on the bias of this solution regarding the previous solution and the current temperature. Regularly, the process starts with a high temperature to allow deterioration in the first iteration. After that, the temperature decreases gradually to demonstrate the evolution of the solution quality. The probability of the worse solutions being accepted is getting smaller. These reflect the idea of initializing sampling the search space in larger steps and then progressively moving towards smaller, more promising areas for the final solution [Riesen et al., 2017]. In other words, it can be considered as the transition process from exploration of possible solutions to the exploitation of more qualitative solutions in the search space. In [Riesen et al., 2017], SA is utilized as a post-processing method to improve the solution of the GED. At the beginning, the lower bound (d_{low}) and upper bound (d_{up}) of the true edit distance between G and H are approximated based on the cost matrix $\mathbf{C}$. Subsequently, the SA is employed to search for a better solution within the interval $[d_{low}, d_{up}]$. SA starts with an initial matching given by the d_{up}. Within the loop, SA attempts to reduce the current upper bound until it equals the predefined d_{low}. Therefore, a temperature decrease factor F is defined in advance. SA stops when it reaches the maximum number of iterations ($it_{\max}$). The inputs contain the cost matrix $\mathbf{C}$ and the starting temperature. The SA-based proposed approach has experimented for univalent GM of undirected graphs. These graphs are built from the datasets (AIDS, fingerprints, GREC) in the IAM graph database [Riesen and Bunke, 2008]. The numerical results show a significant improvement in the solution quality. Additionally, SA is employed for matching shape correspondence in 3D images. [Holzschuh et al., 2020; Ezzati et al., 2021]. In particular, Ezzati et al. investigate SA for the exact GM problem (isomorphism GM). They found that SA is not a good candidate for that problem due to the low chance to get the true matches [Ezzati et al., 2021]. Currently, SA is almost used for solving the univalent GM, i.e., one-to-one correspondence between vertices and edges.

In addition to the mentioned algorithms, there are other meta-heuristics applied for GM problems. In [Bengoetxea et al., 2001; Cesar Jr et al., 2005], EDA is proposed for inexact GM problem. Its principle is similar to GA, but the selection of best matching for the next generation is based on estimation of probability distribution [Cesar Jr et al., 2005].

High order GM has been studied with Particle Swarm Optimization (PSO) [Gong et al., 2016], or Discrete Artificial Electric Field (DAEF) algorithm [Yadav et al., 2020]. Another important meta-heuristic is ACO, the main solver in the thesis for the GM problem, which will be discussed in detail in a separate section afterward.

3.1.4 Other learning-free approaches

3.1.4.1 Bipartite GM heuristic

The bipartite GM is one of the most famous methods for solving the GED problem [Darwiche, 2018]. It is a suboptimal method developed by Riesen et al. [Riesen et al., 2007b]. First, it considers the GED problem as an assignment problem. In particular, from a Quadratic Assignment Problem (QAP), GED is reduced to a Linear Sum Assignment Problem (LASP). The edge matching induced by vertex mappings is neglected. Instead of that, an estimation of edge cost is integrated with the vertex cost to consider the local structures on the considered vertices. Such integrated costs are stored in a particular matrix, called the cost matrix. From that, the Munkres algorithm [Munkres, 1957] is utilized to find the path with the cheapest cost. Formally, these main steps are as follows.

GED as QAP As claimed by Riesen [Riesen, 2015], GED can be formulated as QAP problems, where its complexity is $\mathcal{NP}-$complete. To solve a QAP, one must satisfy two conditions generally: 1) QAP is usually used for sets of equal cardinality; 2) the assignments in QAP are bijective. The author resolves these constraints by augmenting the vertex sets in both graphs with virtual vertices. Specifically, given two attributed graphs $G = (V_G, E_G, \mu_G, \xi_G)$ and $H = (V_H, E_H, \mu_H, \xi_H)$, $|V_G| = m, |V_H| = n$, then $V_G^\varepsilon = V_G \cup \{\varepsilon_1, \cdots, \varepsilon_n\}, V_H^\varepsilon = V_H \cup \{\varepsilon_1, \cdots, \varepsilon_m\}$. So, the extended vertex sets in G and H now have the same cardinality: $(n+m)$. Then, the GED is calculated on the graphs with extended vertex sets, and we assume that $G = (V_G^\varepsilon, E_G, \mu_G, \xi_G)$, and $H = (V_H^\varepsilon, E_H, \mu_H, \xi_H)$. Following that, the adjacency matrices of G and H obtain the same dimensions, or:

$$\mathbf{A} = \begin{array}{c|ccc|ccc} & 1 & \dots & m & 1 & \dots & n \\ \hline 1 & a_{1,1} & \dots & a_{1,m} & \varepsilon & \dots & \varepsilon \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ m & a_{m,1} & \dots & a_{m,m} & \varepsilon & \dots & \varepsilon \\ \hline 1 & \varepsilon & \dots & \varepsilon & \varepsilon & \dots & \varepsilon \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ n & \varepsilon & \dots & \varepsilon & \varepsilon & \dots & \varepsilon \end{array} \quad \mathbf{B} = \begin{array}{c|ccc|ccc} & 1 & \dots & n & 1 & \dots & m \\ \hline 1 & b_{1,1} & \dots & b_{1,n} & \varepsilon & \dots & \varepsilon \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ n & b_{n,1} & \dots & b_{n,n} & \varepsilon & \dots & \varepsilon \\ \hline 1 & \varepsilon & \dots & \varepsilon & \varepsilon & \dots & \varepsilon \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ m & \varepsilon & \dots & \varepsilon & \varepsilon & \dots & \varepsilon \end{array}$$

The elements in matrices $\mathbf{A}$ and $\mathbf{B}$ indicate the edges connecting vertices in G and H, respectively. For instance, if there is an edge between vertices u_i and u_j, a_{ij} will refer to that edge. For finite simple graph, a_{ij} is either 0 or 1. Due to dummy vertices are added in vertex sets of, i.e., V_G^ε and V_H^ε, the value ε in $\mathbf{A}$ and $\mathbf{B}$ represent the relation between a vertex u_i and a dummy vertex. There is no edge in this situation. In other word, ε is found for the infeasible cases [Riesen, 2015].

Cost matrix for GED From these adjacent matrices, the cost matrix $\mathbf{C_{GED}}$ for GED is defined in 3.4. Each block in $\mathbf{C_{GED}}$ represents an edit operation. From left to right, and from upper to lower: vertex substitution with size of (m, n), vertex deletion with size of (m, m), null matrix with size of (n, m), vertex insertion with size of (n, n). At each position (i, j), $c_{i,j}$ denotes the cost of doing one specific operation. For example, $c_{1,1}$ is the cost of substitution vertex $u_1 \in V_G$ to vertex $v_1 \in V_H$. Nevertheless, for deletion and insertion operations, each vertex is deleted or inserted uniquely one time. Therefore, any non-diagonal element of the right-upper part or the left-upper part are set to infinite. For the right-lower part, all costs are set to 0 because operation $(\varepsilon \to \varepsilon)$ should not introduce any cost.

$$\mathbf{C_{GED}} = \begin{array}{c|ccc|ccc} & 1 & \dots & n & \varepsilon_1 & \dots & \varepsilon_m \\ \hline 1 & c_{1,1} & \dots & c_{1,n} & c_{1,\varepsilon} & \dots & \infty \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ m & c_{m,1} & \dots & c_{m,n} & \infty & \dots & c_{m,\varepsilon} \\ \hline \varepsilon_1 & c_{\varepsilon,1} & \dots & \infty & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \varepsilon_n & \infty & \dots & c_{\varepsilon,n} & 0 & \dots & 0 \end{array} \tag{3.4}$$

From adjacency matrices $\mathbf{A}$ and $\mathbf{B}$, and cost matrix $\mathbf{C_{GED}}$ for graphs G and H with extended vertex sets, an optimization problem is stated [Riesen, 2015]:

$$(\varphi_1, \dots, \varphi_{(m+n)}) = \underset{(\varphi_1,\dots,\varphi_{(m+n)})\in\chi_{(m+n)}}{\arg\min} \left[\sum_{i=1}^{m+n} c_{i\varphi_i} + \sum_{i=1}^{m+n}\sum_{j=1}^{m+n} c(e_{ij} \to e_{\varphi_i\varphi_j}) \right], \tag{3.5}$$

where $(\varphi_1, \dots, \varphi_{(m+n)})$ is a permutation, $\chi_{(m+n)}$ is the set of all $(m+n)!$ possible permutations from $1, 2, \dots, (m+n)$. The optimal permutation $(\varphi_1, \dots, \varphi_{(m+n)})$ is equivalent to a bijective assignment from vertices in V_G^ε to vertices in V_H^ε :

$$\lambda = \{u_1 \to v_{\varphi_1}, \dots, u_{(m+n)} \to v_{\varphi_{(m+n)}}\}$$

Hence, this assignment equals to a valid edit path $\lambda \in \Theta(G, H)$ in Def. 13. In Eq. 3.5, there are linear term and quadratic term so that it is a standard QAP. The linear term is the sum of vertex edit costs, the quadratic term is the sum of edge edit costs regarding the permutation $(\varphi_1, \dots, \varphi_{(m+n)})$. It means when a vertex $u_i \in V_G^\varepsilon$ is assigned to $v_k \in V_H^\varepsilon$, and $u_j \in V_G^\varepsilon$ is assigned to $v_l \in V_H^\varepsilon$, then the edge $(u_i, u_j) \in E_G \cup \{\varepsilon\}$ should be associated to edge $(v_k, v_l) \in E_H \cup \{\varepsilon\}$. These edges maybe empty due to additional virtual vertices as shown in $\mathbf{A}$ and $\mathbf{B}$. Now, the GED problem can be calculated under the form of:

$$d_{\lambda_{min}} = \min_{(\varphi_1,\dots,\varphi_{(m+n)})\in\chi_{(m+n)}} \left[\sum_{i=1}^{m+n} c_{i\varphi_i} + \sum_{i=1}^{m+n}\sum_{j=1}^{m+n} c(e_{ij} \to e_{\varphi_i\varphi_j}) \right] \tag{3.6}$$

Reduce GED to a LSAP and solve it by means of Munkres algorithm As indicated in [Riesen, 2015], the formula 3.6 can be simplified to facilitate the computation.

So, the graph distance value $d_\lambda(G,H)$ (not necessary optimal) can be subdivided into two terms corresponding to an edit path λ. That is one term for vertex edit cost, and one term for edge edit costs as in Eq. 3.7, with the assumption that the permutation $(\varphi_1,\ldots,\varphi_{(m+n)})$ and the edit path λ are equivalent

$$d_\lambda(G,H) = C_\lambda^V + C_\lambda^E, \tag{3.7}$$

where $C_\lambda^V = \sum_{i=1}^{m+n} c_{i\varphi_i}, C_\lambda^E = \sum_{i=1}^{m+n}\sum_{j=1}^{m+n} c(e_{ij} \to e_{\varphi_i\varphi_j})$.

In order to employ the Munkres algorithm, [Riesen et al., 2007b] has dropped the quadratic term in Eq. 3.6. Therefore, only the linear term $C_\lambda^V = \sum_{i=1}^{m+n} c_{i\varphi_i}$ is retained. By this reduction, GED is now considered as an LSAP. Nevertheless, if only costs of vertex mappings are considered, it will miss the information on the edges. To strengthen the cost matrix $\mathbf{C_{GED}}$, they also take into account the information on the neighbor of each vertex (adjacent edges). Hence, the local structure of each vertex is integrated with the vertex information. The computation of the local edge costs is as below.

- Vertex substitution $u_i \to v_j$, $(u_i \in V_G, v_j \in V_H)$: let E_{u_i}, E_{v_j} be sets of incident edges of u_j and v_j, respectively. Then, a square edge cost matrix $\mathbf{Ce}$ is built from E_{u_i}, E_{v_j} with respect to the edge cost functions defined in advance. The size of $\mathbf{Ce}$ is $(|E_{u_i}| + |E_{v_j}|) \times (|E_{u_i}| + |E_{v_j}|)$. The Munkres algorithm is also applied to find the minimum sum of edge operation costs [Munkres, 1957], or $c_{i,j} = c(u_i \to v_j) + Munkres(\mathbf{Ce})$
- Vertex deletion $u_i \to \varepsilon(u_i \in V_G)$: deleting a vertex will remove all its adjacent edges, or $c_{i,\varepsilon} = c(u_i \to \varepsilon) + \sum_{e\in E_{u_i}} c(e \to \varepsilon)$
- Vertex insertion $\varepsilon \to v_j(v_j \in V_H)$: inserting a vertex will insert all its adjacent edges, or $c_{\varepsilon,j} = c(\varepsilon \to v_j) + \sum_{e\in E_{v_j}} c(\varepsilon \to e)$

After $\mathbf{C_{GED}}$ is augmented with local edge costs, the Munkres algorithm is used to find the suboptimal solution for the GED problem. This solution is only the vertex-to-vertex mapping between G and H, including deleted and inserted vertices. Then, a full edit path with the vertex-vertex and edge-to-edge mappings is reconstructed. The edge-to-edge operations are induced by the vertex mappings. The final cost is calculated based on the operations selected for vertices and edges, excluding $\varepsilon \to \varepsilon$. The complexity of this algorithm is $O((m+n)^3)$ [Riesen and Bunke, 2009]. Therefore, the computational time is polynomial with the size of $\mathbf{C_{GED}}$.

Thanks to the low complexity of the Munkres algorithm, BP is known to be very fast and can deal with large instances of graphs [Riesen and Bunke, 2009]. BP has been applied in several applications such as handwritten letter recognition, image classification, and biometrics [Riesen et al., 2007b; Riesen and Bunke, 2009]. Despite its strength in execution time, its generality and ability to produce a near-optimal solution are questionable. The reason is that only the local structure is examined rather than a global one. Therefore, other works have proposed an attempt to overcome this limitation by integrating more information linked to vertices [Serratosa and Cortés, 2015; Riesen and Ferrer, 2016], applying more search strategies (greedy swap in [Riesen et al., 2014]). More robust results are presented in these works in comparison with the original BP. Details about the Munkres algorithm can be found in the Appendix B.

3.1.4.2 Iterative search-based approaches

Berretti et al. proposed GED for active GM in image content retrieval [Berretti et al., 2004, 2007]. The vertex merging is augmented besides classical operations to deal with over-segmented and under-segmented images. For each graph derived from each image, the set of vertices is augmented with sets of virtual vertices. These sets consist of adjacent vertices with some shared features. Then, a comparison of two graphs is done between actual-to-actual, actual-to-virtual, and virtual-to-virtual vertices. Therefore, they allow one-to-many, many-to-one, and many-to-many matchings. In addition, they assume that the cost of vertex merging should be less than the costs of vertex insertion and vertex deletion. The solution is constructed iteratively regarding the dissimilarity function of weighted Euclidean distance. Selecting the next candidates is greedy to reduce the complexity of the matching process. As a result, the solution is sub-optimal. The proposed approach is applied for content-based image retrieval. For graph representation, vertices are regions, and edges are region adjacencies. Labels on vertices are region areas and colors. The empirical result shows that activating vertex merging achieves superior performance than inactivating it. In [Berretti et al., 2004], the edge merging induced by vertex merging and its cost are not considered until the later work [Berretti et al., 2007]. The cost of vertex merging relies on the color but not the regional area feature. Besides, computing the set of virtual vertices for each vertex results in a significant increase in the computational time. Finally, the proposed method is not compared with other methods.

3.1.4.3 Graph embeddings

In addition to the previous methods, graph matching can be solved by graph embedding techniques, which embed the vertices in a graph into points in a low-dimensional geometric space. As a result, the GM problem is transformed from mapping of vertices and edges into matching the weighted distributions of points in the normed spaces [Demirci et al., 2006]. Then, a measure function is utilized to search for the optimal matching between distributions. Earth mover distance algorithm is an example [Demirci et al., 2006, 2011]. Finally, the result is projected to the initial vertex matching for graphs. One of the advantages of this technique is that it is tractable for comparing large graphs [Demirci et al., 2006]. Nevertheless, the graph structure is somehow not reversed due to the embedding process of vertices. Normally, the numerical labels are put on the vertices, and only weighted edges are allowed. Therefore, it is not practical for graphs with symbolic attributes. Only the vertices and their labels are considered after the embedding process. Consequently, the edge relation induced by the vertex matching is not studied. Furthermore, the edge construction after matching result is usually ignored [Darwiche, 2018]. In [Demirci et al., 2006, 2011], many-to-many matching is considered for 2D and 3D object recognition from datasets, such as Coil-20, ETH-80, Rutgers Tool Database in [Demirci et al., 2006], MPEG-7 CE-Shape-1 [Demirci et al., 2011]. The graphs are derived from the shock tree (directed or undirected). Vertices are shock points, and edges are adjacent shocks with a weight indicating Euclidean distance between shock points. In [Kpodjedo et al., 2013], many-to-many GM with embedding technique is used for matching diagrams in design comparison. The TS is used to solve the GM problem. And the fitness function takes into account both vertex and edge relations. However, defining edge relations induced

by vertex merging and splitting is not explicit. Fig. 3.6 gives primary ideas on how to solve GM with the embedding technique.

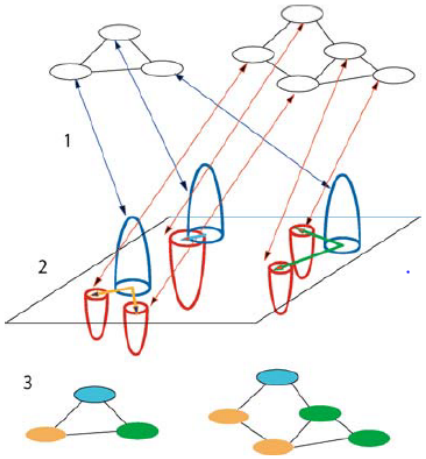


Figure 3.6: An example for many-to-many GM: 1) two initial graphs are embedded into a normed space, with point mass (size of a hole) is a function of a vertex's labels, 2) the points are matched by using earth mover distance algorithm, 3) the vertex-to-vertex correspondence between the original graphs. Image is taken from [Demirci et al., 2006].

3.1.5 Learning graph matching: a focus on the deep learning techniques

In recent years, the development of Deep Learning (DL) has allowed for learnable feature extraction by cascading differentiable layers in flexible architectures [Yan et al., 2020]. As a result, DL has been applied in numerous domains such as computer vision (object tracking, 3D action recognition), telecommunication (networking, resource management) [He et al., 2021; Vesselinova et al., 2020]. On the other hand, Combinatorial Optimization (CO) is an indispensable research direction [Yan et al., 2020]. The combination between DL and CO has led to a remarkable impact on fundamental research and real application. Following this tendency, more and more learning techniques have been used for GM problems. That is because it will be less dependent on the expertise, more generic, more scalable, even more accurate [Yan et al., 2020]. The traditional methods, such as Bipartite GM [Riesen et al., 2007b], are called *learning-free methods* to distinguish from the learning-based methods. In general, the learning-based methods try minimizing the error rate on the number of corrected vertex mapping between two graphs. In other words, it aims to maximize the vertex mappings that are close to the GT.

Formally, given two attributed graphs $G = (V_G, E_G, \mu_G, \xi_G)$ and $H = (V_H, E_H, \mu_H, \xi_H)$, the GM problem seeks an optimal assignment vector $\hat{\mathbf{v}}$ as in 1 [Xu et al., 2021]:

Problem 1 (Graph matching problem)

$$\begin{aligned} & \hat{\mathbf{v}} = \arg\max_{\mathbf{v}} S(G, H, \mathbf{v}) \\ \textit{subject to} \quad & \mathbf{v}_{i,k} \in \{0,1\}^{|V_G|.|V_H|} \\ & \sum_{u_i \in V_G} \mathbf{v}_{i,k} \leq 1, \forall u_k \in V_H \\ & \sum_{u_k \in V_H} \mathbf{v}_{i,k} \leq 1, \forall u_i \in V_G \end{aligned}$$

where $S(G, H, \mathbf{v})$ measures the similarity of graph attributes, i.e., the attributes of mapped vertices and edges. [Xu et al., 2021]. $S(G, H, \mathbf{v})$ can be decomposed into a first order similarity function $s(u_i, u_k)$ for a vertex pair $u_i \in V_G$ and $u_k \in V_H$ and a second order similarity function $s(e_{ij}, e_{kl})$ for an edge pair $e_{ij} \in E_G$ and $e_{kl} \in E_H$. The second term is quadratic and thus S is a quadratic function regarding variable $\mathbf{v}$. In the form of vectorization, S can be expressed as:

$$S(\mathbf{v}, \mathbf{sv}, \mathbf{se}) = \mathbf{sv}^T \cdot \mathbf{v} + \sum_{e_{ij} \in E_G} \sum_{e_{kl} \in E_H} \mathbf{v}_{i,k} \cdot \mathbf{v}_{j,l} \cdot \mathbf{se}_{ij,kl} \tag{3.8}$$

where $\mathbf{sv}$ represents the vertex similarity $\{s(u_i, u_k) | u_i \in V_G, u_k \in V_H\}$, and $\mathbf{se}$ stores the edge similarity $\{(e_{ij}, e_{kl}) | e_{ij} \in E_G, e_{kl} \in E_H\}$.

Given a graph pair, L measures the difference between the predicted matching ($\hat{\mathbf{v}}$) and the ground truth one ($\mathbf{v}^{gt}$). Then, the learning GM problem can be stated as minimizing the average loss function on a dataset of graph pairs. Formally, the learning GM problem is defined as follows [Xu et al., 2021].

Problem 2 (Learning graph matching problem)

$$\hat{\theta} = \arg\min_{\theta} \sum_{((G,H),\mathbf{v}^{gt}) \in TrS} L\left(\mathbf{v}^{gt}, \Phi(\theta; G, H)\right) \tag{3.9}$$

where $\Phi : G \times H \times \theta \rightarrow \hat{\mathbf{v}}$ is a mapping function, θ are trainable parameters (invariants) over the training dataset TrS, and L is the loss function. A pipeline demonstrating the supervised learning of GM problem is given in Fig. 3.7. Inside the problem of learning GM, three main sub-problems are arisen as in Fig. 3.8.

Types of learning are various, but they are divided into two main sub-categories: shallow methods and deep methods [Raveaux, 2019]. The first category concentrates on learning the cost functions for vertex and edge matchings. It can be minimized or maximized regarding the cost functions predefined. For example, Cho et al. have adopted structured SVM for a unified framework to learn the specific weight of each vertex and edge, rather than consistent weight allocation in learning-free methods [Cho et al., 2013]. Besides the structure learning-based methods, other researchers propose to learn GM based on network embedding or graph embedding techniques, which embeds the graph information into a continuous vertex representation. These methods are more efficient to deal with large-scale GM [Yan et al., 2020]. For instance, Du et al. suggested an embedding technique

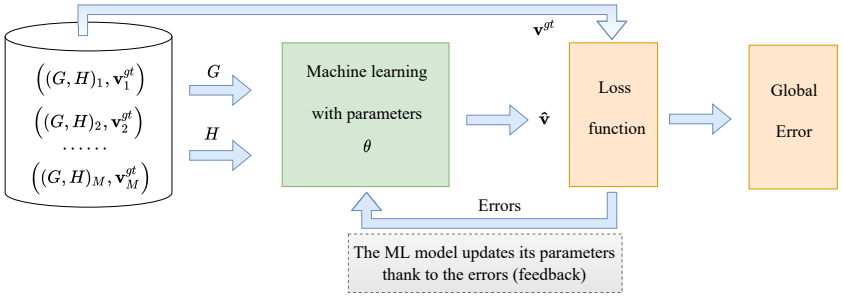


Figure 3.7: A general pipeline for the supervised learning of GM problem. The figure is derived from [Raveaux, 2019].

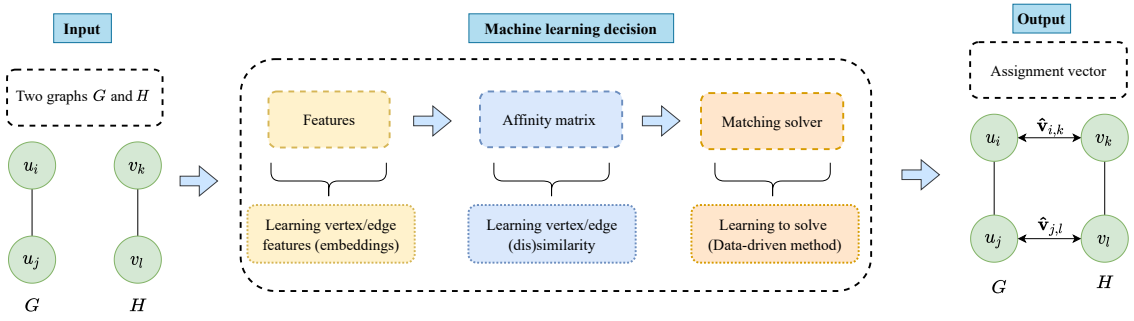


Figure 3.8: Three sub-problems are arisen from learning the GM problem: learning features, learning dissimilarity, and learning to match. The figure is modified from [Raveaux, 2019].

relied on a cross-graph random walk for GM without any seed alignment. Therefore, the method is fully unsupervised [Du et al., 2019]. Overall, the shallow methods focus on optimizing only the cost functions for vertices and edges in a supervised or unsupervised manners. In addition, the graph representation is needed in advance [Raveaux, 2019]. For DL methods, one different point is that they usually propose an end-to-end framework for solving GM. That means they can perform from graph representation to GM [Yan et al., 2020]. Also, a Neural Network (NN) architecture trained by gradient descent with the forward pass and backward pass are defined to learn the GM problem. We will focus more on the DL methods in this section.

Here, we present how to learn the GM problem with a NN integrated with a Graph Matching Solver (GMS) as in Def. 19 [Xu et al., 2021].

Definition 19 (Learning GM problem with a NN)

$$\theta^* = \arg\min_{\theta} \sum_{((G,H),\mathbf{v}^{gt}) \in TrS} L(\mathbf{v}^{gt}, GMS(NNF(G,H,\theta))) \tag{3.10}$$

where

- θ are parameters of NN;

- *NNF* is a NN function such that $NNF(G, H, \theta) \rightarrow [\mathbf{sv} \in \mathbb{R}^{|V_G|\cdot|V_H|}, \mathbf{se} \in \mathbb{R}^{|E_G|\cdot|E_H|}]$;
- **sv** and **se** are predefined vertex and edge similarities, respectively;
- *GMS* is a GM solver such that $GMS : [\mathbf{sv}, \mathbf{se}] \rightarrow \hat{\mathbf{v}}$.

The resolution to this learning problem is finding where the derivative equals 0. Therefore, learning the GM problem with a NN is equivalent to the gradient-based problem as follows [Xu et al., 2021].

Definition 20 (Learning GM problem as a gradient-based problem)
Find θ^* *such that*

$$\sum_{((G,H),\mathbf{v}^{gt})\in TrS} \frac{\partial L\left(\mathbf{v}^{gt}, GMS\left(NNF(G, H, \theta^*)\right)\right)}{\partial \theta^*} = 0 \tag{3.11}$$

To achieve the Eq. 3.11, the gradient descent methods can be used with the update procedure defined as follows:

$$\theta^{t+1} = \theta^t - lr. \sum_{((G,H),\mathbf{v}^{gt})\in TrS} \frac{\partial L\left(\mathbf{v}^{gt}, GMS\left(NNF(G, H, \theta)\right)\right)}{\partial \theta}$$

where $lr.$ is a learning rate that controls the speed of the descent.

Due to L is a function of GMS and NNF as in 3.10, the gradient of the loss according to θ is defined as in Eq. 3.12 by the chain rule.

$$\frac{\partial L(\theta)}{\partial \theta} = \frac{\partial L(\mathbf{v})}{\partial \mathbf{v}} \cdot \frac{\partial GMS(\mathbf{sv}, \mathbf{se})}{\partial [\mathbf{sv}, \mathbf{se}]} \cdot \frac{\partial NNF(\theta)}{\partial \theta} \tag{3.12}$$

This step is done in the backward pass or *back-propagation*.

In order to do the back-propagation in deep GM, the GMS should be differential. Therefore, two main ways to tackle the deep GM problem are the continuous relaxation-based approach, and the combinatorial-based approach [Raveaux, 2019; Yan et al., 2020].

The very first work proposed to marry the shallow (GM) to the DL based on the continuous relaxation is of [Zanfir and Sminchisescu, 2018]. In this work, they showed how to do the forward pass and backward pass of the GM layer in the DL context. The QAP is relaxed to the continuous domain as presented in the sub-section 3.1.2. The power iteration method is utilized to solve the relaxed problem with $\mathbf{v}_{t+1} = \mathbf{K}\mathbf{v}_t / \|\mathbf{K}\mathbf{v}_t\|$, where $\mathbf{K}$ is an affinity matrix defined as in subsection 3.1.2. A similar direction is followed by Wang et al. [Wang et al., 2019]. The GM problem is reduced to a relaxed Linear Sum Assignment Problem (LSAP) where the second-order in Eq. 3.8 is dropped, and $\mathbf{v} \in [0, 1]$. The relaxed LSAP is a special case of optimal transport problem, and it can be solved efficiently with the Sinkhorn algorithm [Altschuler et al., 2017]. All methods proposed in these works are differentiable to do backward pass later. Nevertheless, the combinatorial aspect of the

original GM problem is sacrificed, which can lead to a loss of relations between the local features [Xu et al., 2021].

To avoid losing the combinatorial aspect, [Rolínek et al., 2020] proposed a GM solver based on the Lagrangean decompositions [Vlastelica et al., 2019]. In this manner, the GM problem is split into multiple small sub-problems linked together by Lagrangean multipliers. Thanks to the work of [Vlastelica et al., 2019], the integration of this GM solver is possible. It embedded the blackbox implementation of a wide range of combinatorial algorithms into deep networks in a mathematical manner. Nevertheless, the proposed method yields a sub-optimal solution for the GM problem. Following this concept, [Xu et al., 2021] suggested an exact GM solver integrating into the deep GM. Therefore, the solution is optimal. It is also the first time an exact GM solver is combined with a DL context, aiming to evaluate the impact of GM solvers on deep GM. In detail, the exact mixed integer linear programming solver is used for finding the optimal assignment vector $\hat{\mathbf{v}}$. The back-propagation is realized by a continuous interpolation of the loss function. Their experiments on datasets, Pascal VOC and SPair-71K, show that deep GM should be relaxed carefully to avoid losing solution quality.

Although deep GM has been just developing in recent years, numerous works propose various techniques to tackle the GM problems. The applications are in different domains, such as few-shot learning for 3D action recognition [Guo et al., 2018], entity alignment between knowledge graphs [Fey et al., 2020], pose estimation [Sarlin et al., 2020], or multiple object tracking [He et al., 2021]. One advantage of deep GM is that all steps are done in an end-to-end fashion, i.e., from graph generation to GM resolution. That helps to reduce the reliance on the specialists, and be more generic, more adaptive, and reusable [Yan et al., 2020]. Generally, deep GM-related works propose to solve GM in an approximate way, i.e., the optimal solution is not guaranteed [Nowak et al., 2018; Fey et al., 2020; He et al., 2021; Wang et al., 2019]. Rare works deal with exact GM. To our best of knowledge, only the work of [Xu et al., 2021] uses an exact GM solver for a deep GM architecture. The authors would like to explore the impact of GM solvers and the types of relaxations on the GM results. They think the findings can provide scientific guidance for a better design of deep GM models.

To the best of our knowledge, all of the mentioned works just deal with univalent GM (one-to-one correspondence), and no work considers the multivalent GM problem. On the one hand, learning GM by deep architectures is just emerging recently due to the complexity of graph structure [Bronstein et al., 2017]. On the other hand, multivalent GM is just only at its very early stage [Raveaux, 2019]. Therefore, deep GM matchings are just applied for univalent GM. More efforts should be done to understand more and apply more techniques to solve the multivalent GM. In addition, the mentioned research works usually utilize vertex features for doing GM tasks, and few works integrate the edge labels for solving the GM. Recently, only the edge relations are considered, not the edge labels. The vertex labels are regularly numeric, not symbolic [Raveaux, 2019]. If the edge labels are taken into account, it might increase the accuracy. Also, deep GM requires a large amount of data (samples) for training and testing. That condition might not be satisfied for many specific datasets of many GM problems that are very limited in the number of samples [Abu-Aisheh, 2016].

3.2 A focus on Ant Colony Optimization

From the previous sections, we see that there are many techniques to solve the GM problems. Meta-heuristic algorithms are one of the popular approaches. As aforementioned in subsection 3.1.3, the meta-heuristic algorithms are a good compromise between discrete and continuous optimization techniques. Besides, it is well-known for solving assignment problems, especially QAP [Achary et al., 2021]. GM problem is also an assignment problem. Particularly, GM equals to a QAP [Raveaux, 2019]. Therefore, meta-heuristics are possible for the GM problem. Among meta-heuristic algorithms, we decide to focus on ACO in this thesis due to the following reasons:

- ACO is not specific so that it can be adapted for GM problem
- ACO is a global search heuristic so it can get a more qualitative solution
- ACO can be easy to implement and to parallel
- ACO is one of the most well-known meta-heuristics with solid theoretical background
- Compared with other meta-heuristic algorithms, ACO is the most promising meta-heuristic for QAP according to a recent study of [Achary et al., 2021]
- Review works on solving multivalent GM by meta-heuristic approaches, ACO is one of the regular methods (see subsection 3.1.3). Perhaps that implies a superior performance of ACO on more complicated GM problems than other methods.

These reasons show us the flexibility and the efficiency of ACO in comparison with other methods. Therefore, this thesis gives an attention to ACO for multivalent GM. The following section will explain in more detail the ACO: origin, properties, main works related to GM, and main variants.

3.2.1 General concept of ACO meta-heuristics

Ant Colony Optimization (ACO) was firstly introduced in 1992 in the Ph.D. thesis of [Dorigo, 1992]. From that time, it is exploited in numerous combinatorial optimization problems in different domains. Until now, ACO is one of the most well-known meta-heuristic algorithms [Dorigo and Stützle, 2019]. ACO is a nature-inspired algorithm that simulates the behavior of natural ants. In nature, ants are working in a herd. They communicate together by a chemical substance called *pheromones*. Based on the pheromone, the ants can find the most appealing trails between their nests and food sources. In addition, the pheromone is evaporated over time. Therefore, if a trial is not reinforced with a new deposit of pheromone, it will disappear or be un-appealing. In other words, there is indirect communication among ants [Monmarché et al., 2010].

To replicate the process of the ant colony, a very famous example is given in [Goss et al., 1989]. Given the ants' nest and a food source, there are many paths to go from the nest to the food source. The question is which path is the best (shortest path) in terms of the total distance. The illustration is given in Fig. 3.9. In the figure, there

are two paths where the one from a to d and b is the shortest one. By experimental results, the researchers have these observations. In the beginning, all artificial ants are uniformly distributed. That means they can follow any existing paths in the experiment. Then, by pheromone communication, more and more ants follow the shorter path. So, the pheromone on this path is higher than another path. After a longer time, other ants also take this path and that shows the highest pheromone concentration. The longer path has a very low pheromone concentration due to the pheromone evaporation process. In the end, most ants focus on the shortest path as expected. From the experiment, they discover these main points of a self-organized system [Monmarché et al., 2010]:

- a positive reinforcement mechanism: more pheromone indicates better solution quality
- a negative feedback mechanism: pheromone evaporation limits the pheromone, and it might allow the existence of the stable state and the memory lapse
- a random behavior: this implies that the process is non-stationary and there may have fluctuations in states of the system
- a multiplicity of interactions: the ant colony is numerous.

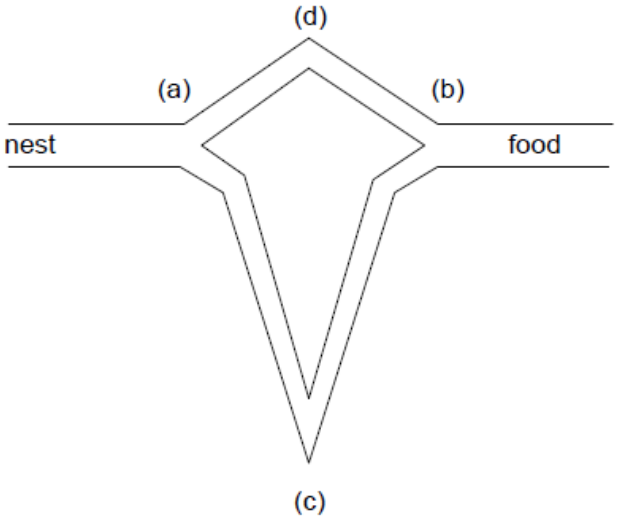


Figure 3.9: Example to simulate the ants' behavior. There are two bridges to connect the nest and the food source. Clearly, there is one path that is much shorter than another regarding the total distance. The figure is taken from [Monmarché et al., 2010].

Although inspiring from the real ants, the ACO metaheuristic needs some essential factors to facilitate the computational process. Those factors are partially or not related to the real ants [Monmarché et al., 2010]. Specifically,

- Ants have memory: the ant has to remember the partial path that it has crossed. That aims to prevent the visited parts, whereas that is not necessary for real ants.

- Pheromone: the artificial ant lays pheromone after building a complete path. The shorter path is, more pheromone are laid. In real life, ants deposit pheromone more regularly and independently on the total length of the path.
- Speed: artificial ant's speed is not constant. It moves between different parts/positions in one unit of time. This attempts to ease the simulation process of ant's movements. After each iteration, the reinforcement is implemented to reward this synchronous simulation and it is proportional to the solution quality.
- Artificial ants are not blind: When ant moves, it takes into account the distance between positions. The distance is the heuristic information that helps ants to better select the next vertices. Moreover, the pheromone amount is also another factor that influences on the decision of ants to construct the path. So, paths built by artificial ants rely on two factors: heuristic and pheromone.

From now on, we call artificial ants by ants in short.

3.2.2 Main variants of ACO

Before introducing the main variants of ACO, we present a typical example of assignment problem-the Traveling Salesman Problem (TSP). Then, main variants of ACO are used in the context of solving TSP. Nevertheless, we are always aware of the connection between the TSP and GM. For TSP, the main target is visiting all the cities with the shortest path (distance) [Lawler, 1985]. These cities can be considered as vertex associations in GM, and the purpose of GM is matching vertices in two compared graphs with the smaller cost [Sammoud et al., 2005]. The edge mappings are implied by the vertex matching. Therefore, there are some common points between TSP and GM. Particularly, both of them are well-known combinatorial problems. However, more constraints can be put on the GM to adapt with different types of GM problems.

TSP: a typical example of assignment problem Traveling Salesman Problem (TSP) is a classical combinatorial problem, firstly formulated in 1930 [Lawler, 1985]. Given a list of cities and the distances between every two cities, one needs to find the shortest path such that: a) visit all the cities; b) without repeating any visited cities; c) return to the initial city. Therefore, the tour is a Hamiltonian cycle. Although the problem seems easy to understand and solve, its complexity is $\mathcal{NP}$-hard in CO. This problem is very famous in theoretical computer science and operational research. Hence, it is usually used to test proposed optimization methods [Dorigo and Stützle, 2019].

The TSP has several applications in numerous domains, such as planning and logistics [Laporte et al., 1996]. Depending on the context and the problem, the concept of 'city' can represent different things. Due to the reputation of TSP, many algorithms have been designed to solve it efficiently. Among them, ACO is one of the promising candidates for TSP [Dorigo and Stützle, 2019]. Originally, ACO is proposed to solve TSP [Dorigo, 1992]. Therefore, we will use the TSP to explain the main variants of ACO metaheuristics in later parts. First, we give a simple formulation for the TSP problem.

Definition 21 (TSP) *Let $d(i,j)$ be the distance from city i to j, then, the TSP tries to search a shortest tour $\mathcal{T}^*$ among possible tours $\mathcal{T}$ such that:*

$$L(\mathcal{T}^*) = \min_{\mathcal{T}} \left\{ \sum_{i=1}^{n-1} d(\mathcal{T}_i, \mathcal{T}_{i+1}) + d(\mathcal{T}_n, \mathcal{T}_1) \right\} \tag{3.13}$$

where $L(\cdot)$ is length of a valid tour.

3.2.2.1 Ant System

The Ant System (AS) algorithm is associated with the Travelling Salesman Problem (TSP). It is proposed to solve this problem, and it is also the first variant of ACO [Dorigo, 1992]. But, it is not the most efficient variant. It is often used as a Benchmark to verify new ideas and algorithmic variants [Dorigo and Stützle, 2019].

Building a solution Within an iteration, each ant starts with an empty tour $\mathcal{T} = \{\}$. It constructs the tour incrementally. The first city is added randomly because the tour is a Hamiltonian cycle. From a current city i, the ant selects the next candidate to add to the partial tour based on the transition probability as in Eq. 3.14. It repeats this step until the tour is a Hamiltonian cycle (including all the cities).

$$Pr_{ij}^k(it) = \frac{\tau_{ij}^\alpha(it) \times \eta_{ij}^\beta}{\sum_{l \in \Xi_i^k} \tau_{il}^\alpha(it) \times \eta_{il}^\beta} \tag{3.14}$$

where

- η_{ij} presents the visibility of the ant or the heuristic information
- Ξ_i^k are the cities that have not been visited by ant k from the city i
- α and β weight the relative impact between pheromone and heuristic factors.

In the Eq. 3.14, the heuristic information is often calculated with respect to the distance between two cities, i.e., $\eta_{ij} = 1/d_{ij}$. That means two closer cities giving better heuristic value. The denominator in Eq. 3.14 is to normalize the probability.

Pheromone update After all ants complete constructing tours at one iteration, the pheromone update is performed. For AS, each ant k deposits a pheromone amount Δ_{ij}^k on each edge of its tours, i.e., $e_{ij} = (i,j) \in \mathcal{T}^k(it)$. This amount is proportional to the length of the tour, or

$$\Delta_{ij}^k(it) = \begin{cases} 1/L(\mathcal{T}^k(it)) & \text{if} \quad (i,j) \in \mathcal{T}^k(it) \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

After that, the pheromone on each edge (i,j) is updated as follows:

$$\tau_{ij}(it+1) = (1-\rho) \times \tau_{ij}(it) + \sum_{k=1}^{pop} \Delta_{ij}^k(it) \tag{3.16}$$

where ρ is the evaporation rate, $\rho \in [0, 1]$.

The algorithm 1 presents an entire framework for AS.

Algorithm 1 General framework of Ant System algorithm

```
Pheromone initialization τ_ij ← τ_0, i, j = 1, ··· , n
for it ← 1 to it_max do
    for k ← 1 to pop do
        Build a tour T^k(it)
        Compute the length of tour L(T^k(it))
    end for
    Update T*
    for each edge (i, j) do
        Update pheromone τ_ij as in Eq. 3.16
    end for
end for
return shortest tour found T*
```

3.2.2.2 Max-Min Ant System

Max-Min ant System (MMAS) is another variant of ACO which was developed by [Stützle and Hoos, 2000]. This variant is a development from AS with several improvements, particularly in the pheromone management process. First of all, the pheromone is bounded in an interval, i.e, $\tau_{ij} \in [\tau_{\min}, \tau_{\max}]$. This constraint is made to assure a good balance of pheromone on the most visited edges and the least visited ones. It also gives a chance for infrequently edges to diverse the searching process. So, it can avoid being trapped in local optima. After, the pheromone update process is more greedy than AS. It updates pheromone only on the best ant's tour of the current iteration or on the best-so-far tour from the beginning. That leads to a faster solution convergence. Finally, the pheromone initialization is set to $\tau_{\max}$. In case of stagnation of the search process, the pheromone may be reinitialized to explore better the search space.

Building a solution The process of building a solution for each ant is similar to that of AS.

Pheromone update Pheromone update of MMAS includes two main steps: update and adjustment.

Firstly, the pheromone is brought up date as in Eq. 3.17.

$$\tau_{ij}(it + 1) = (1 - \rho) \times \tau_{ij}(it) + \Delta_{ij}^{+} \tag{3.17}$$

where $\Delta_{ij}^{+} = 1/L^{+}$, L^{+} is the length of either *best-iteration tour* ($\mathcal{T}_{itbest}$) or *best-so-far tour*

($\mathcal{T}_{bestsofar}$).Then, all the pheromone is adjusted in bound $[\tau_{\min}, \tau_{\max}]$ as in Eq. 3.18.

$$\tau_{ij} = \begin{cases} \tau_{\min} & \text{if} \quad \tau_{ij} < \tau_{\min} \\ \tau_{\max} & \text{if} \quad \tau_{ij} > \tau_{\max} \end{cases} \tag{3.18}$$

The algorithm 2 presents an entire framework for Max-Min Ant System.

Algorithm 2 General framework of Max-Min Ant System

Pheromone initialization $\tau_{ij} \leftarrow \tau_{\max}, i, j = 1, \cdots, n$
for $it \leftarrow 1$ to it_{max} **do**
  for $k \leftarrow 1$ to *pop* **do**
    Build a tour $\mathcal{T}^k(it)$
    Compute the length of tour $L(\mathcal{T}^k(it))$
    Perform a local search (optional)
  end for
  Update $\mathcal{T}^*$
  for each edge (i, j) **do**
    Update pheromone τ_{ij} as in Eq. 3.17
    Adjust pheromone values as in Eq. 3.18
  end for
end for
return shortest tour found $\mathcal{T}^*$

3.2.2.3 Ant Colony System

Ant Colony System (ACS) is another subsequent variant of ACO, after AS and MMAS. Its initial target is also proposed to deal with TSP [Dorigo and Gambardella, 1997]. In comparison to AS and MMAS, it is more dissimilar both in solution construction and pheromone update. For each of them, two levels are integrated: local and global. With that combination, it is thus well-known as one of the best performing ACO variants [Monmarché et al., 2010] for the TSP problem.

Building a solution During one iteration, each ant constructs a Hamiltonian cycle like in previous variants. However, it selects the next candidate to add to the partial tour by choosing exploitation or exploration strategies. This decision is made regarding the value of a random uniform value (q) and the predefined threshold q_0 ($q, q_0 \in [0, 1]$). More details are given in Table 3.1.

Pheromone update The pheromone updating process of this variant includes two steps: local and global as follows.

Local update At each ant step, when an ant goes from i to j, the local pheromone update is done on each edge. More precisely, the pheromone on each edge is evaporated an

Strategy	Condition	Formula
Exploitation	$q \leq q_0$	$j = \arg\max_{l \in \Xi_i^k} \left\{ \tau_{il} \times \eta_{il}^{\beta} \right\}$
Exploration	$q > q_0$	$Pr_{ij}^k = \frac{\tau_{ij} \times \eta_{ij}^{\beta}}{\sum_{l \in \Xi_i^k} \tau_{il} \times \eta_{il}^{\beta}}$

Table 3.1: The ant selects a candidate j from city i either by exploiting or exploring respecting the value of threshold q_0 and uniform random number q. Value of q changes at each step to choose candidates. Other notations are defined similarly as in previous sections.

amount of $(1-\xi)\tau_{ij}$, and augmented an amount of $\xi\tau_0$ as in Eq. 3.19. Here, ξ is the local pheromone evaporation. This idea is different from the previous variants and somewhat misleading. However, this scheme is used to balance the pheromones on high and low visited edges. When numerous ants use the same edge, the pheromone on this edge will converge to the initial pheromone τ_0. So, it helps to reduce the pheromone bias between edges and to increase the diversity in the solution building process.

$$\tau_{ij} = (1-\xi)\tau_{ij} + \xi\tau_0. \tag{3.19}$$

Global update The global pheromone update is somehow similar to the MMAS variant. It is the elitist scheme where only edges on the best tour (best-iteration or best-so-far tour) receives an additional pheromone respecting its solution quality. However, only those edges have pheromone evaporation as in Eq. 3.20. That is another different point of the ACS variant. In that way, it considers only edges belonging to L^+, not all edges. Therefore, it is more time-saving in this step in comparison with other variants.

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho\frac{1}{L^+}, \forall (i,j) \in \mathcal{T}^+, \mathcal{T}^+ \in \{\mathcal{T}_{itbest}, \mathcal{T}_{bestsofar}\} \tag{3.20}$$

The algorithm 3 presents a general framework for ACS.

3.2.3 ACO for graph matching problems

As one of the member of meta-heuristic algorithms, ACO is known to be efficient for solving the combinatorial optimization problem [Dorigo and Stützle, 2019]. GM is also a complex combinatorial problem, and several works apply ACO for tackling it. In this section, we review some works that utilize ACO for the GM problems (univalent, multivalent, and others). Furthermore, our main studied problem is multivalent GM. Therefore, we go deeper into works addressing such problem. Finally, a summary is given to have an overview on the mentioned works.

3.2.3.1 ANT-GM for multivalent GM

The first work using ACO for multivalent GM that can be mentioned is of [Sammoud et al., 2005]. The proposed algorithm is named ANT-GM. Based on the similarity measure

Algorithm 3 General framework of ACS

Pheromone initialization $\tau_{ij} \leftarrow \tau_0, i, j = 1, \cdots, n$
for $it \leftarrow 1$ to it_{max} **do**

  for $k \leftarrow 1$ to *pop* **do**
    Build a tour $\mathcal{T}^k(it)$ following rules in Table 3.1
    Compute the length of tour $L(\mathcal{T}^k(it))$
    Perform a local search (optional)
    Perform a local pheromone update as in Eq. 3.19
  end for

  Update $\mathcal{T}^*$

  Update pheromone on L^+ as in Eq. 3.20

end for
return shortest tour found $\mathcal{T}^*$

proposed by [Champin and Solnon, 2003], the authors introduced a new ACO for comparing the multi-labeled graphs. Promising results are obtained with ACO in comparison with greedy and reactive tabu approaches on subgraph isomorphism problems and multivalent GM ones. Specifically, the detail of ANT-GM is below.

Multi-labeled graphs by [Sammoud et al., 2005] A multi-labeled graph is a triple $G = (V, r_V, r_E)$, where:

- V is a finite set of vertices,
- $r_V \subseteq V \times L_V$ is a relation that assigns labels in to each vertex, i.e., $r_V = \{(v_i, l)\}$ such that $v_i \in V$ is labeled by $l \in L_V$,
- $r_E \subseteq V \times V \times L_E$ is a relation that assigns labels to edges, i.e., $r_E = \{(v_i, v_j, l)\}$ such that edge $(v_i, v_j) \in V \times V$ is labeled by $l \in L_E$.

According to [Sammoud et al., 2005], a multi-labeled graph is a *directed graph*. And the tuples r_V, r_E are the features of graph G. The multi-labeled graphs defined above are different from the labeled graph defined in Definition 2 because the multi-labeled graph is dedicated for directed graphs. Therefore, the labeled graphs in Definition 2 is more general. The authors define $descr(G) = r_V \cup r_E$ is the set that describes completely the graph G. Based on $descr(G)$, the similarity measure is given subsequently.

Similarity measure The similarity measure originated in [Champin and Solnon, 2003] is proposed for evaluating the similarity between two multi-labeled graphs $G = (V_G, r_{V_G}, r_{E_G})$ and $H = (V_H, r_{V_H}, r_{E_H})$. The measure is defined over the same sets of vertex and edge labels L_V and L_E such that $V_G \cap V_H = \emptyset$. Before measuring the graph similarity, the vertex

mapping is built. The matching here is multivalent, each tuple $(u, v) \in V_G \times V_H$ signifies a vertex u matched to the vertex v. From the multivalent matching m, the set of common features are calculated based on $descr(G)$ and $descr(H)$ as follows:

$$\begin{aligned} descr(G) \cap_m descr(H) = &\{(u, l) \in r_{V_G} | \exists (u, v) \in m, (v, l) \in r_{V_H}\} \\ &\cup \{(v, l) \in r_{V_H} | \exists (u, v) \in m, (u, l) \in r_{V_G}\} \\ &\cup \{(u_i, u_j, l) \in r_{E_G} | \exists (u_i, v_k) \in m, \exists (u_j, v_l) \in m, (v_k, v_l, l) \in r_{E_H}\} \\ &\cup \{(v_k, v_l, l) \in r_{E_H} | \exists (u_i, v_k) \in m, \exists (u_j, v_l) \in m, (u_i, u_j, l) \in r_{E_G}\} \end{aligned}$$

Because m is a multivalent matching, we need to identify the set of split vertices as well. This set contains vertices matched to more than one vertex in the other graph. Given s_u is the set of vertices that a vertex u is matched with, then:

$$\begin{aligned} splits(m) = &\{(u, s_u) | u \in V_G, s_u = \{v \in V_H | (u, v) \in m, |s_u| \geq 2\} \\ &\cup \{(v, s_v) | v \in V_H, s_v = \{u \in V_G | (u, v) \in m, |s_v| \geq 2\} \end{aligned}$$

Then, the similarity between G and H regarding matching m is defined by:

$$sim_m(G, H) = \frac{f(descr(G) \cap_m descr(H)) - g(splits(m))}{f(descr(G) \cup descr(H)} \tag{3.21}$$

where f, g are two functions that weigh the features and splits, being determined by specific applications. The denominator is just for normalizing. Therefore, it is sufficient to optimize the score function with respect to each matching m, i.e.,

$$score_m = f(descr(G) \cap_m descr(H)) - g(splits(m)) \tag{3.22}$$

So, the target is to find the matching m that maximizes the score between two graphs. Henceforth, the score function can be referred to the fitness function. Then, the authors propose a new ACO algorithm for the GM problem, called ANT-GM. The algorithm follows a classical ACO algorithmic scheme for a static combinatorial optimization problem. There are four main steps for the ANT-GM: the construction graph, the pheromone trails, construction of a matching, and pheromone update.

Construction graph for ANT-GM Given two multi-labeled graphs $G = (V_G, r_{V_G}, r_{E_G})$ and $H = (V_H, r_{V_H}, r_{E_H})$, the construction graph G_{ants} is a complete undirected graph for ants. Vertices in G_{ants} are couple of vertices in G and H, i.e., $(u, v) \in V_G \times V_H$. In addition, ants lay pheromone trails on G_{ants}. Ants select couples of vertices in G_{ants} to build the matching m iteratively.

Pheromone trails In this work, pheromone trails are laid on the edges of the construction graph. A pheromone laid between couples of vertices (u_i, v_k) and (u_j, v_l) is denoted $\tau_{\langle (ui, v_k), (u_j, v_l) \rangle}$. This pheromone amount reflects the learnt desirability of matching u_i with v_k, u_j with v_l simultaneously.

Construction of a matching by one ant At each iteration, each ant starts with an empty matching, i.e., $m = \{\}$. Then, couples of vertices are selected to add into the current matching m incrementally. These couples are taken from the candidate set $cand = \{(u, v) \in V_G \times V_H - m\}$. The decision of choosing the next couple to add into m relies on the transition probability, i.e.,

$$Pr_m(u, v) = \frac{[\tau_m(u, v)]^{\alpha}.[h1_m(u, v)]^{\beta_1}.[h2_m(u, v)]^{\beta_2}}{\sum_{(u',v')\in cand}[\tau_m(u', v')]^{\alpha}.[h1_m(u', v')]^{\beta_1}.[h2_m(u', v')]^{\beta_2}} \tag{3.23}$$

where

- $\tau_m(u, v)$ is the pheromone factor. It equals to sum of all pheromone trails laying between the candidate (u, v) and the selected couples (u', v') in m, i.e., $\tau_m(u, v) = \sum_{(u',v')\in m} \tau_{\langle(u,v),(u',v')\rangle}$,
- $h1_m(u, v)$ is the first heuristic factor. It favors candidates that increase most the score function, i.e., $h1_m(u, v) = score(m \cup \{(u, v)\}) - score(m)$
- $h2_m(u, v)$ is the second heuristic factor. It favors couples that have many looked-ahead features, i.e., $h2_m = f(look_ahead_m(u, v))$, where $look_ahead$ is a set that includes incident edges of u and v which have not been in m yet.
- α, β_1 and β_2 weight the relative importance of three factors.

New couples are added to the current matching m until one of these stopping conditions are satisfied:

- there is no couple that can increase the score function;
- the $look_ahead$ set is empty, i.e., there is no common edge features in $look_ahead$;
- the score function has not been increased since the last three moves.

Pheromone update Pheromone is updated after all ants have finished constructing their matchings at the current iteration. There are two main processes: pheromone evaporation and pheromone reinforcement. The evaporation process reduces the pheromone on each edge by an amount of $(1-\rho)$, where ρ is the evaporation rate, $\rho \in [0, 1]$. The reinforcement process augments the pheromone on edges belonging to the best path of the iteration m_k. The augmented pheromone amount is $1/(1 + score(m_{best}) - score(m_k))$, where m_{best} is the best matching from the beginning (including the current iteration).

The proposed ANT-GM is tested on subgraph isomorphism problems and multivalent GM problems. Its numerical results are compared with greedy and Reactive Tabu Search (RTS) approaches. Overall, the greedy approach is less robust than ANT-GM and RTS for two considered problems. For multivalent GM problems, RTS is relatively more robust than ANT-GM in terms of execution time and accuracy. In general, ANT-GM and RTS

are very complementary, each approach can solve instances that another cannot solve. [Sammoud et al., 2005].

Overall, it is encourageable that Sammoud et al. applied ACO for multivalent GM problems for the first time [Sammoud et al., 2005]. Although promising numerical results are obtained, there are some considerations. Firstly, the similarity proposed in [Champin and Solnon, 2003] can only work with directed graphs and with discrete attributes. Consequently, it restricts the range of applications. Secondly, the defined matching m presents uniquely the common parts of two graphs, not the difference such as unmapped vertices/edges. Next, setting parameters for α, β_1, β_2 is not explicit. There are a high bias between value of α versus values of β_1 and β_2, i.e., 1 versus 8, and 3, respectively. Finally, the experiments for the multivalent GM problem are done uniquely for unlabeled graphs. Therefore, we observe only the impact of topology on the score function, not the labels.

3.2.3.2 ANT-GM'06 and ANT-GM'06+LS

Continuing this approach, the authors proposed later an augmented version of ANT-GM called ANT-GM'06, and ANT-GM'06+LS in [Sammoud et al., 2006]. First, another heuristic function is proposed to compute the transition probability. Next, the authors considered a new pheromone strategy. Finally, a Local Search (LS) is integrated to enhance the solution constructed by ants. The augmented versions without LS, with LS, are named ANT-GM and ANT-GM'06+LS, respectively. Thanks to these improvements, more robust numerical results are observable in the experiments for multivalent GM problems. Below, we present the mentioned modifications for ANT-GM'06 and ANT-GM'06+LS.

Pheromone trails In [Sammoud et al., 2006], the pheromone trails are laid on the vertices of the construction graph G_{ants}. Therefore, the pheromone amount on each vertex $(u, v) \in G_{ants}$ represents the learnt desirability to match u with v when constructing the matching. Furthermore, laying pheromone trails on the vertices of G_{ants} has linear complexity with respect to the number of vertices in G_{ants}. As a result, it is more time-saving than laying the pheromone trails on the edges of G_{ants} in [Sammoud et al., 2005]. The pheromone on a vertex of G_{ants} is denoted $\tau(u, v)$.

Construction of a matching by one ant The process of constructing one matching by one ant is similar to the previous work [Sammoud et al., 2005]. However, there are some modifications to compute the transition probability. Specifically, only one heuristic factor is taken into account rather than two. More formally, given a matching m and a set of candidates $cand = \{(u, v) \in V_G \times V_H - m\}$, the probability $Pr_m(u, v)$ to select the couple (u, v) is:

$$Pr_m(u, v) = \frac{[\tau(u, v)]^{\alpha}.[\eta_m(u, v)]^{\beta}}{\sum_{(u', v') \in cand}[\tau(u', v')]^{\alpha}.[\eta_m(u', v')]^{\beta}} \tag{3.24}$$

where $\tau(u, v)$ is the pheromone value on vertex $(u, v) \in G_{ants}$; $\eta_m(u, v)$ is defined similarly to $h1_m(u, v)$ in Eq. 3.23; α and β weight the relative importance of the pheromone and heuristic factors.

Ants stop adding new candidates to the matching m when one of these conditions is satisfied: 1) adding that candidate decreases the score function; 2) when the score function has not been increased since the last three moves.

Pheromone update The ANT-GM'06 follows the scheme of Max-Min Ant System (MMAS) [Stützle and Hoos, 2000]. Hence, the initial pheromone is set to $\tau_{\max}$. The pheromone update includes pheromone evaporation, pheromone reinforcement, and pheromone adjustment. First, pheromone on each vertex of G_{ants} evaporates an amount of $(1 - \rho)$, with ρ aforementioned. Then, best ant of the iteration m_k deposits pheromone on each couple of vertices in m_k. The deposited pheromone quantity equals to $1/[1 + (score(m_{best}) - score(m_k))]$, where m_{best} is the best matching from the beginning, included the current iteration. Finally, all vertices that have the pheromone amount outside the range of $[\tau_{\min}, \tau_{\max}]$ will be adjusted in that range.

Local search procedure To enhance the matching quality, a LS search is integrated with ANT-GM. The selected LS is said to have a good compromise between the quality and computational time [Sammoud et al., 2006]. Formally, the best matching of the iteration-m_k is enhanced by LS. The three worst couples in m_k are excluded, and replaced by other couples in a greedy way, i.e., incrementally adding couples that increase most the score function. This process is repeated until there is no further improvement found. Remark that the removed couples are the couples in m_k that have not been removed yet.

The improved versions of ANT-GM, ANT-GM'06, and ANT-GM'06+LS, are tested on two types of GM problems. The first one is on multivalent GM problems with directed unlabeled graphs. It is a synthetic dataset. The graphs contain a maximum of 100 vertices and around 360 edges. The second one is on the non-bijective GM problems proposed in [Boeres et al., 2004] for scene recognition. The graphs are directed and labeled, containing a maximum of 250 vertices. This problem is mainly one-to-many matching because it aims to associate between the model image with their over-segmented images [Boeres et al., 2004]. The numerical results show that ANT-GM'06+LS is more robust than ANT-GM'06 thanks to the LS integration. It is pretty competitive in terms of solution quality between ANT-GM'06+LS and RTS. For more complicated instances in the second case, ANT-GM'06+LS generates better matching than RTS. Nevertheless, RTS is much faster than ANT-GM'06+LS in all cases. Although ANT-GM is relatively inferior to RTS in terms of computational time, it is superior to RTS regarding solution quality for more complicated instances. Therefore, it can be employed for complex problems if the time constraint is more relaxed. Anyway, the employed similarity measure deals with only labels of discrete values. Hence, it may be a problem when tackling the graphs with continuous labels on vertices/edges. Additionally, the edge relations here are not mentioned just for substitution

cases, not for other cases. Besides, there is still a big bias between the values of α and β. For example, $\beta = 10$, meanwhile $\alpha = 1$ for the first experiment on multivalent GM problems. The unbalance proportion seems that the heuristic factor is far more important than the pheromone factor in ANT-GM's versions. Finally, only quantitative GM results are analyzed. But, it is also essential to study the GM results at a low level, i.e., sub-part correspondence.

3.2.3.3 Other ACO utilisation for solving GM problems

In addition to ACO-based algorithms for multivalent GM from the group of Sammoud et Sorlin [Sammoud et al., 2005, 2006; Sorlin and Solnon, 2005; Sorlin et al., 2007], there exists numerous works solving other types of GM problems with ACO. We present some works for univalent GM, high-order GM, and multiple GM as follows.

ACO for univalent GM In [Ding et al., 2014], Ding et al. used ACO for detecting Maximum Common subgraph (MCS) in retrieving 3D CAD. The authors propose a Hierarchical Graph (HG) which describes better a 3D CAD model. Its vertices are features of CAD modeling, and its edges are dependency relations between parts of the model. Moreover, HG is a directed tree. It describes not only shape features on vertices but also the topology relationship on edges. A generic ACO is integrated into GA to tackle the MCS problem. GA is responsible for searching for a sub-optimal solution first, then ACO improves it to achieve an optimal solution. Therefore, each ant starts with an initial matching produced from GA. The pheromone is laid on the vertices of the construction graph, and only the best solution at the current iteration will deposit an extra pheromone on its path. The proposed method is tested on the real dataset named National Design Repository with 700 CAD models for coarse and fine-grain retrievals. More robust results are obtained with GA-ACO against GA alone. Nevertheless, parameter setting for GA and ACO is not presented. Only the pheromone factor is employed for improving the sub-optimal solution from GA.

Another work of Gong et al. proposed to use ACO, called AntMapper, for map matching, i.e., mapping raw GPS trajectories onto the digital road network [Gong et al., 2017]. A road network is a directed graph where its vertices are road intersections, and its edges are road segments in the map. According to the authors, it is the first work that considers using ACO for map matching. The ants work on a construction graph to search for the best mapping. Its vertices represent candidate matched points given as input of the problem. Pheromones are put on the edges of the construction graph. The ACO's mechanism in this paper looks similar to ACS where local and global pheromone updates are implemented. For global update, only the best-so-far ant will deposit an additional pheromone on its path. The algorithm stops when it reaches the maximum number of iterations. The proposed ACO is implemented on synthetic and real datasets containing 10 and 40 vertices, respectively. Promising results are obtained, but the algorithm does not take all types of road segmentation into account. Moreover, the proposed method is just suitable for off-line usage of trajectories such as urban planning, traffic dynamic analysis. The GM problem considered here is error-tolerant subgraph isomorphism.

ACO for high-order GM In [Wu et al., 2019], Wu et al. proposed using ACO for solving high-order GM. Instead of considering first-order (vertex relations) and second-order (edge relations), high-order GM utilizes the high-order relations to build the correspondences between two or more sets of features [Chertok and Keller, 2010]. Specifically, the triplet of vertices (*hyper-edges*) are considered as a high-order relation in this work. Given two graphs $G = (V_G, E_G), H = (V_H, E_H)$, the high-order GM is formulated as:

$$S(\mathbf{P}) = \sum_{i,k,j,l,t,h} \mathbf{T}_{i,k,j,l,t,h} \mathbf{P}_{i,k} \mathbf{P}_{j,l} \mathbf{P}_{t,h}$$

where $\mathbf{T}$ is the 6D supersymmetric tensor, $\mathbf{T}_{i,k,j,l,t,h}$ is the affinity between two hyper-edges of two graphs, i.e., $e^G_{i,j,t} = (u_i, u_j, u_t) \in V_G$ and $e^H_{k,l,h} = (v_k, v_l, v_h) \in H$, respectively; $\mathbf{P}$ is a binary permutation matrix. $S(\mathbf{P})$ is the objective function that maximizes the sum of the affinity values of all matched hyper-edges. To impose the *one-to-one correspondence*, the assignment matrix $\mathbf{P}$ has to satisfy: $\sum_i \mathbf{P}_{i,j} \leq 1, \sum_j \mathbf{P}_{i,j} \leq 1$. Then, the ACO is employed to construct the matching such that $S(\mathbf{P})$ is maximized. The pheromone is laid between pairs of vertices on the construction graph defined similarly as in [Sammoud et al., 2005]. The pheromone reinforcement is performed locally and globally. On the local scale, all ants depose an amount of pheromone on their matchings. On the global scale, only the best ant so far is allowed to deposit an additional pheromone. The pheromone evaporation is done on each pair of vertices in the construction graph. The proposed algorithm experiments on the synthetic-CMU house and natural datasets for matching tasks. However, labels on vertices and edges of the graph representations for these datasets are not mentioned, and graphs are undirected. The studied GM problem is error-correcting graph isomorphism. Therefore, both graphs have the same number of vertices. The experiments show that the proposed method is more robust, yet more time-consuming than the Particle Swarm Optimization (PSO)-based methods.

ACO for multiple GM Multiple GM means matching of more than two graphs at the same time [Ha et al., 2013; Schiff et al., 2018]. Normally, the solution of the GM contains pairs of vertices. For multiple GM, the solution contains a $n-$tuple of vertices representing n graphs matched together. For instance, the solution of a triple matching is a set $S = (u, v, z)$, where $u \in G1, v \in G2$, and $z \in G3$ [Schiff et al., 2018]. There are works of Ha et Xuan, and Schiff that propose ACO for solving multiple GM [Ha et al., 2013; Schiff et al., 2018]. In [Ha et al., 2013], more than three graphs are matched together in the context of comparing biological molecules. ACO is used to find the solution to the problem by maximizing the score function. This objective function is composed of the scores of vertex and edges. The construction graph is built as the search space for ants. The graph contains n layers corresponding to match n graphs simultaneously. The ith layer denotes the vertices in the ith graph G . An ant searches a matching (solution) by starting from the first vertex in the first layer to reach the last layer and inversely. It is repeated until all vertices of all graphs are included in the matching. The GM problem is similar to a GED, but its cost functions are not explicitly explained. The MMAS variant is used in this work. The one-to-one correspondence is imposed for vertex mappings. The proposed approach is tested only on a synthetic dataset. The graph representation is not explicit, with no mention of labels of vertices/edges. The maximum graph order is 50.

Experimental results show the superior performance of the proposed approach compared with the GA and greedy approaches.

Another work is of Schiff for triple GM [Schiff et al., 2018]. Similarly, the one-to-one correspondence constraint is imposed for vertex matching. Each ant begins with an empty 3D permutation array M. The ant adds a candidate (u, v, z) to M at each iteration if any vertex in the triple has not been selected. The object is maximizing the number of matchings in the solution. No heuristic information is used, and only pheromone is used to enhance solutions in the next iterations. The maximal graph order is 50 vertices. No information about the labels of vertices/edges is mentioned, and the GM type is not specified.

For a summary of works using ACO for the GM problem, one can refer to Table 3.2. One may find out that most of the works consider one-to-one correspondence from ordinary GM to high-order GM and multiple GM. Only a few studies deal with multivalent GM despite its generality. According to [Raveaux, 2019], multivalent GM is just at its early stage, and more works are necessary to understand it more deeply.

Reference	GM problem	ACO's variant	OTO	Graphs	Remarks
[Ding et al., 2014]	univalent	generic	yes	directed, labeled	no heuristic information for ACO no parameter study for ACO
[Gong et al., 2017]	univalent	ACS	yes	directed, labeled	off-line usage only solution quality influenced by initial candidate sets
[Sammoud et al., 2005]	multivalent	generic	no	directed	score function just works with symbolic labels
[Sammoud et al., 2006]	multivalent	generic	no	directed	similarly
[Sorlin et al., 2007]	multivalent	generic	no	directed, labeled	similarly
[Wu et al., 2019]	high-order	generic	yes	undirected	no mention about labels of vertices, edges high computational time
[Ha et al., 2013]	multiple GM	MMAS	yes	not precise	no mention about labels of vertices, edges high computational time
[Schiff et al., 2018]	multiple GM	generic	yes	not precise	not precise the dataset

Table 3.2: Summary of some related works using ACO for GM problems from traditional to multiple GM, from univalent to multivalent GM., OTO refers to one-to-one mapping.

3.3 Conclusion on the state-of-the-art

In this chapter, we have presented the main techniques to solve the GM problems. They are tree search, continuous relaxation, meta-heuristic, deep learning, and graph embedding. No method certainly is the best one. Each has its advantages and limitations. The tree search can solve the GM problem in an exact or inexact way. However, it requires memory and time to traverse the search tree to achieve a qualitative solution. Besides, the selection of good heuristic functions to estimate the upper and lower bounds influences significantly

the solution quality [Raveaux, 2019]. Therefore, it may not be a good choice for solving large GM. For continuous relaxation, it is more time-saving than tree search, but the solution is regularly sub-optimal due to the relaxation process [Conte et al., 2004]. Also, many methods are influenced by the initialization step of solution [Conte et al., 2004]. The meta-heuristics can be considered as a good balance between tree search and continuous relaxation [Laura et al., 2021]. It can be easily adapted to solve various types of GM problems. Nevertheless, they are non-deterministic methods that give different solutions at running times. Consequently, it is better to run several times to ensure the confidence of the solution [Raveaux, 2019]. Deep learning shows its robustness by having the capacity to implement GM in an end-to-end fashion. Moreover, it can learn both graph representation and graph mapping functions to optimize the accuracy. Therefore, it is more generic and less dependent on the expert knowledge [Yan et al., 2020]. The main problem of DL methods comes from the need of huge amount of training data, which is not usually available. As a consequence, it has been applied on restricted range of applications such as image matching. In addition, the mentioned research works usually utilize vertex features for doing GM tasks, and few works integrate the edge labels for solving the GM. Recently, only the edge relations are considered, not the edge labels. The vertex labels are regularly numeric, not symbolic [Raveaux, 2019]. Graph embedding can tackle large graphs by projecting vertices into a Euclidean space. Nonetheless, the edge relation is not considered during the matching process. At a different point of view, it is more a matching of points rather than GM [Conte et al., 2004].

During the reviewing process, we note that these techniques are mostly applied for univalent matching, i.e., one-to-one correspondence. Some works deal with multivalent GM, i.e., one vertex can be associated with more than one vertex in another graph, but they are restricted to very specific applications. That shows the multivalent is just at its early step where more studies are vital to deal with it more [Raveaux, 2019]. As stated in chapter 2, we focus on the multivalent GM, and we heed to formulate it as an extended GED. Among the reviewed GM techniques mentioned above, we see that meta-heuristics are good candidates to tackle the multivalent GM due to the following reasons:

- Meta-heuristics are a good balance between the tree search and the continuous relaxation [Laura et al., 2021];
- Deep learning-based techniques are data-driven method [Yan et al., 2020], but to the best of our knowledge, no work has been found to solve multivalent GM because of limited data for this GM problem;
- Meta-heuristics are well-known for solving the complex assignment problem such as TSP, QAP [Dorigo and Stützle, 2019; Achary et al., 2021]
- Meta-heuristics are able to give a good solution in a reasonable time [Blum and Roli, 2003]
- Meta-heuristics are used in a lot of works to solve the multivalent GM (one-to-many, many-to-one, many-to-many) as presented in subsection 3.1.3

Among meta-heuristic methods utilized for multivalent GM, we found that ACO, GA, SA, and TS are the most popular methods for solving the GM problems. However, SA

is proved to be inefficient for tackling the exact GM [Ezzati et al., 2021]. Also, no work employs SA for multivalent GM to the best of our knowledge. ACO, GA, and TS are utilized for tackling the multivalent GM. However, TS is a local search, so it has a higher risk to be stuck in a local solution [Bianchi et al., 2009; Yang, 2011]. In contrast, ACO and GA are global search methods that are more efficient to search for the global optimum [Yang, 2011]. Between ACO and GA, some studies show superior performances of ACO against GA [Achary et al., 2021]. On the one hand, GM is equivalent to a QAP [Raveaux, 2019]. On the other hand, the recent work that evaluates the performance of meta-heuristics (ACO, GA included) for QAP shows that ACO obtains the best results [Achary et al., 2021]. Besides, there exists other works solving multivalent GM by ACO [Sammoud et al., 2005, 2006; Sorlin et al., 2007]. However, their proposed similarity measure is merely designed for directed graphs with symbolic attributes. As a consequence, it restricts the range of applications. Also, the comparison between ACO's variants is not presented. Hence, we believe that there is still room to explore ACO's capacity for solving multivalent GM. Specifically,

- Design a general pipeline of ACO for tackling multivalent GM;
- Evaluate the performance of ACO's variants for the multivalent GM;
- Propose different strategies to solve multivalent GM with ACO;
- Analyse the influence of ACO's parameters to the multivalent GM;
- Study the order of the graph in which ACO is still able to process the multivalent GM;
- Is ACO better than other meta-heuristics for multivalent GM;
- Compare robustnesses of multivalent GM and univalent to answer the question: univalent or multivalent GM, which one is better and in which context?

Part II

ACO for solving multivalent GM based on a GED formulation

Chapter 4

From multivalent GM to Extended GED

Contents

4.1 Formulate multivalent GM in the form of ExGED 85
4.2 Reformulation of cost matrix for ExGED 86
4.3 Illustrative example 98
4.4 Summary 103

Abstract
In the previous chapter, we have given a state-of-the-art on the multivalent GM and methods for solving this problem. We observe that meta-heuristic methods can be good candidates for solving multivalent GM. Especially, ACO is one of the most promising meta-heuristic methods that achieve many successes in different domains, including combinatorial problems. Therefore, in this chapter, we try the first step to combine multivalent GM and ACO. To facilitate that, we reformulate the problem as an extended GED. Then, we apply ACO for solving the reformulated problem. A cost matrix is constructed as input for the solver to exploit the heuristic information. In addition, we also provide an analysis of the cost matrix in terms of size and computation. We realize that the cost matrix, in this case, is more complicated than GED. Hence, some assumptions are made to simplify the cost matrix and the calculation. Finally, we present an example to demonstrate the construction and computation of the cost matrix in detail.

4.1 Formulate multivalent GM in the form of ExGED

On the one hand, according to the literature review, few works deal with multivalent GM. Among them, there is the work of Sammoud et al. [2005] that defines multivalent GM as a relation between sets of vertices. Nevertheless, the edge relation is not considered in this definition. For other works, multivalent GM is defined in a general way like one vertex in the source graph can be mapped to more than one vertex in the target graph Ambauen et al. [2003]; Boeres et al. [2004]; Deruyver et al. [2005]. On the other hand,

most works study incomplet variants of GED-based methods for solving multivalent GM. Therefore, it is better to give a formal problem formulation before solving it. The definition of multivalent GM in the form of GED can be more convenient. There exist numerous works studying techniques for tackling GED. Expressing multivalent GM as an Extended GED will help us to involve both vertex and edge relations. Further, the edit operations in GED allow us to be more flexible to adapt the ExGED regarding the problem considered. So, those are our motivations to formulate the multivalent GM in the form of ExGED.

From Def. 13, the classical edit operations allow only univalent mapping. The splitting and merging operations are supplemented to consider multivalent mapping. Then, the ExGED is defined as follows.

Definition 22 (ExGED) *Given two attributed graphs* $G = (V_G, E_G, \mu_G, \xi_G)$ *and* $H = (V_H, E_H, \mu_H, \xi_H)$*, we define:*

1. *merging the set* $S_{mer} = \{u_i \in V_G, i \geq 2\}$ *to* $v \in V_H$ *is noted* $S_{mer} \to v$
2. *splitting a vertex* $u \in V_G$ *into the set* $S_{spl} = \{v_j \in V_H, j \geq 2\}$ *is noted* $u \to S_{spl}$
3. $(S_{mer} \to v) \wedge (u \to S_{spl}) \wedge (u \to v) = \emptyset$, $\forall u \in V_G, v \in V_H, S_{mer} \subset V_G, S_{spl} \subset V_H$
4. *ExGED is also a dissimilarity measure derived from the costs of edit operations*

$$d_{min}(G, H) = \min_{\lambda \in \Theta(G,H)} \sum_{ed_i \in \lambda} c(ed_i) \tag{4.1}$$

where $ed_i \in \{substitution, deletion, insertion, splitting, merging\}$

From Def. 22, the 1st and 2nd items state two additional operations apart from classical ones in Table 2.1. The 3rd item implies the constraint of GED that each vertex/edge is assigned to exactly one edit operation. The 4th item shows that ExGED is also a dissimilarity measure. Therefore, the target is to search for an edit path with a minimum cost. Table 4.1 displays the difference between GED and ExGED regarding the availability of edit operations. The properties of this new formulation will be mentioned in a separate section afterward. In addition, the operations on edges induced from vertex operations will be in another part. Even in GED, the metric property is also hard to respect for some problems. This condition might lead to a less robust performance to some extent in the analysis of Serratosa [2019]. As a consequence, we will not pay much attention to this aspect for ExGED. But, ExGED is always non-negative as GED.

4.2 Reformulation of cost matrix for ExGED

4.2.1 A non-square cost matrix for ExGED

Regularly, a cost matrix is built for GED, and then an algorithm is used to find the optimal (suboptimal) solution according to this matrix Riesen [2015]. For example, Riesen et al. construct a square cost matrix for GED and employ the Hungarian algorithm to solve it Riesen et al. [2007b]. It is noticed that the cost matrix of GED should be square

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

Table 4.1: Availability of edit operations for GED and ExGED (with $u \in V_G, v \in V_H, e_G \in E_G, e_H \in E_H, \varepsilon$ is the *virtual* vertices or edges). S_{mer} and S_{spl} are subsets of V_G and V_H, respectively as in Def. 22. Edge split and merge will be detailed in a separate section 4.2.2.2.

to use the Hungarian algorithm. Nevertheless, this constraint is not consistent if other solvers are utilized Serratosa [2014, 2015]. For our proposed approach, the cost matrix is not obligatorily square.

Moreover, the cost matrix plays a vital role in dealing with GED because it defines the goodness of mappings between vertices and edges Riesen [2015]. For ExGED, it is similar. A cost matrix that contains good heuristic information should be provided to the solver. Furthermore, storing heuristic information in a matrix will reduce the memory and accelerate the computational process. In that way, no need to recompute everything from the beginning when constructing the edit paths.

The initial idea for the cost matrix will follow the form in Eq. 3.4 with splitting and merging in ExGED. Thus, additional blocks are added to represent these operations. There is only one row and one column for virtual vertices. Given $G = (V_G, E_G, \mu_G, \xi_G)$ and $H = (V_H, E_H, \mu_H, \xi_H)$ two attributed graphs, $P^k_{mer} = \{S^i_{mer}\}$ be the set of all possibilities for merging of vertices in G, and $P^k_{spl} = \{S^j_{spl}\}$ be the set of all possibilities for splitting vertices in G, k is a parameter that allows maximum number of vertices that one vertex can be matched with. Then k is a very important parameter when considering multivalent matching and will be strongly discussed in the following sections. Note that merging vertices in G is equivalent to splitting a vertex in H and inversely. Denote that $m = |V_G|, n = |V_H|, h = |P^k_{mer}|$, and $l = |P^k_{spl}|$, then the cost matrix for ExGED is defined below.

$$\mathbf{C} = \begin{array}{c|cccc|ccc} & 1 & \dots & n & \varepsilon & 1 & \dots & l \\ \hline 1 & c_{1,1} & \dots & c_{1,n} & c_{1,\varepsilon} & c_{1,S^1_{spl}} & \dots & c_{1,S^l_{spl}} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ m & c_{m,1} & \dots & c_{m,n} & c_{m,\varepsilon} & c_{n,S^1_{spl}} & \dots & c_{m,S^l_{spl}} \\ \hline \varepsilon & c_{\varepsilon,1} & \dots & c_{\varepsilon,n} & 0 & \infty & \dots & \infty \\ \hline 1 & c_{S^1_{mer},1} & \dots & c_{S^1_{mer},n} & \infty & \infty & \dots & \infty \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h & c_{S^h_{mer},1} & \dots & c_{S^h_{mer},n} & \infty & \infty & \dots & \infty \end{array} \tag{4.2}$$

From Eq. 4.2, two edit operations are supplemented. The splitting operations which locate in the upper-right block refer to each vertex in V_G being split into a set of vertices S^j_{spl}, i.e., $u \to S^j_{spl}, u \in V_G$. Inversely, it can be considered as merge on the graph H. The merging operations which locate in the lower-left block refer to sets of vertices in $S^i_{mer} \subset V_G$ being merged into one vertex in V_H, i.e., $S^i_{mer} \to v, v \in V_H$. Reversibly, merging on G equals splitting on H. To be simple, we consider splitting and merging from graph G towards graph H. In addition, it forbids the mappings between the virtual vertices and sets of vertices for splitting and merging. So, the costs from these mappings will be set to infinite. Also, the cost matrix $\mathbf{C_{ExGED}}$ is not a square matrix as $\mathbf{C_{GED}}$ in 3.4. That is because the sizes of splitting and merging sets can be different, i.e., $h \neq l$. The size of $\mathbf{C_{ExGED}}$ is $(m+1+h) \times (n+1+l)$, which shows that $\mathbf{C_{ExGED}}$ is much bigger and more difficult to deal with than $\mathbf{C_{GED}}$.

4.2.2 An insight into cost matrix for ExGED

4.2.2.1 Dealing with the cost matrix size

From cost matrix in Eq. 4.2, we remark these main points:

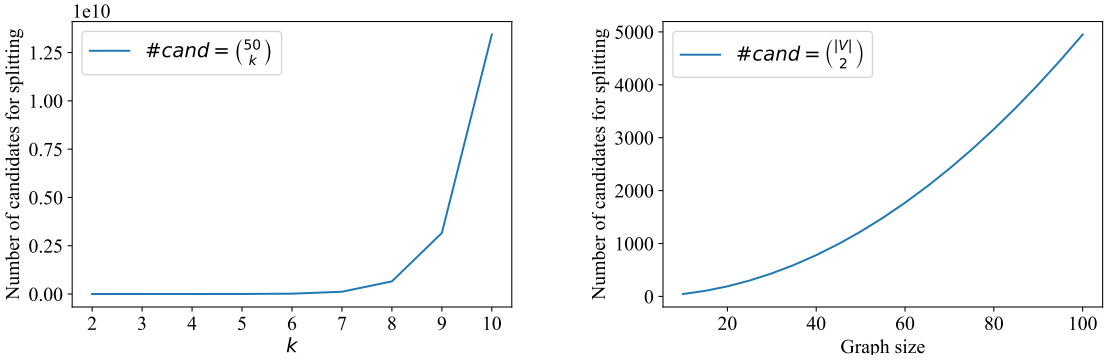
- matrix $\mathbf{C_{ExGED}}$ is not always a square matrix
- size of $\mathbf{C_{ExGED}}$ depends initially on the size of graphs G and H
- size of $\mathbf{C_{ExGED}}$ depends on the cardinalities of sets P^k_{mer} and P^k_{spl}
- sets P^k_{mer} and P^k_{spl} are influenced by parameter k, the maximum size of split/merge set.

Hence, size of $\mathbf{C_{ExGED}}$ is determined not only by graphs G and H, but also by the sets P^k_{mer} and P^k_{spl}. The orders of graphs G and H are unchangeable, but sizes of sets P^k_{mer} and P^k_{spl} are changeable regarding k. So, the orders of initial graphs G and H are not the size of $\mathbf{C_{ExGED}}$ as in the case of $\mathbf{C_{GED}}$. Notably, k has an impact on the cost matrix. That is because k decides the size of extended sets for splitting and merging. Therefore, we will do some numerical simulations to demonstrate this influence.

First of all, it is about the *impact of k on the cardinality of splitting/merging sets.* Given two graphs G and H with the same size, i.e., $|V_G| = |V_H| = |V| = 50$, we suppose

that the maximum number of vertices that one vertex can match with varies from 2 to 10, i.e., $k \in [2, 10]$. Notice that when $k = 2$, it means that one vertex can be mapped to at most two other vertices. However, when $k = 4$, all the possibilities for splitting/merging will be an accumulated sum. For instance, if $k = 4$, it is probable that one vertex can be split to two vertices, three vertices, or four vertices, i.e., $|P_H^k| = \sum_{k=2}^{4} \binom{|V|}{k}$. Indeed, $\sum_{k=0}^{n} \binom{n}{k} = 2^n$. As a consequence, $|P_H^k|$ will grow exponentially as in Fig. 4.1i when k increases.

Next, we investigate the case where k is fixed to 2-the minimal value, and the graph size rises gradually. It aims to observe how the number of splitting/merging candidates augments despite the minimal value of k. As shown in Fig. 4.1ii, when the graph is larger, the number of splitting/merging candidates increases exponentially. That implies that given a higher value of k, the candidates for merging/splitting will be enormous for graphs from 60 vertices.



(i) k changes, $|V| = 50$ (ii) $k = 2$, $|V|$ change

Figure 4.1: Influence of k on the number of candidates for splitting in different cases.

Finally, we consider directly the influence of k on the size of $\mathbf{C_{ExGED}}$. To do that, we suppose that two graphs having the same size $m = n = |V| = 10$. We give the value of k in range of $[2, 5]$. For each value of k, we compute:

- number of rows: $nb_rows = m + n + \binom{n}{k}$
- number of cols: $nb_cols = m + n + \binom{m}{k}$
- all elements of $\mathbf{C_{ExGED}}$: $nb_rows \times nb_cols$

Fig. 4.2 demonstrates the increase of size of $\mathbf{C_{ExGED}}$ regarding values of $k \in [2, 5]$. Similar to previous investigations, the size of $\mathbf{C_{ExGED}}$ grows remarkably even if k is not very big.

From these examples, we see that k plays a key role in constructing the cost matrix $\mathbf{C_{ExGED}}$. Therefore, selection of k is really important in dealing with multivalent matching. Moreover, it is better to impose some constraints for vertices that can be merged or split. That can reduce the size of $\mathbf{C_{ExGED}}$. This selection relies on the expert-knowledge to some extent.

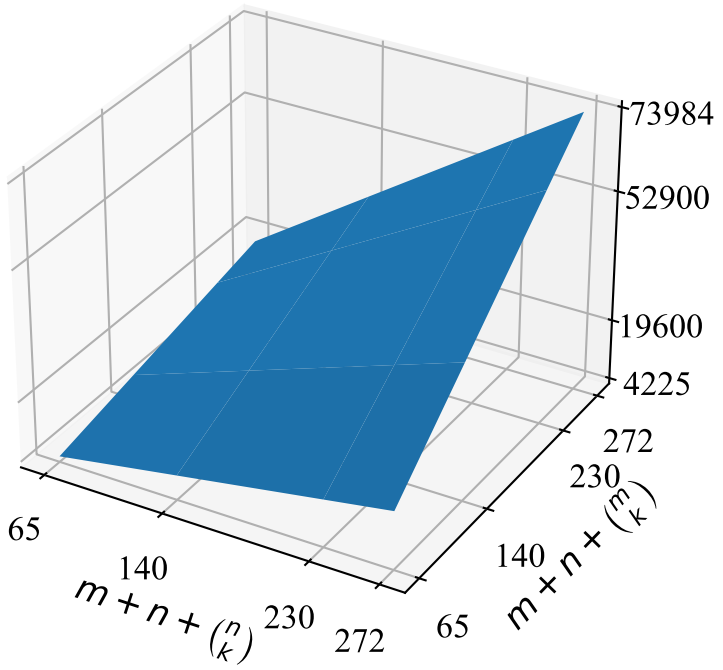


Figure 4.2: Impact of k on the size of the cost matrix $\mathbf{C_{ExGED}} = \left(m+n+\binom{n}{k}\right) \times \left(m+n+\binom{m}{k}\right)$, given $k = 2,3,4,5$; m, n are numbers of vertices in two graphs.

4.2.2.2 Dealing with the edge relations for merging and splitting

In addition to the size of $\mathbf{C_{ExGED}}$, how to define the edge relations when having splitting/merging is also a tricky question. In the literature review, most of the works avoid this question or just mention it shortly (see Table 2.4). In this section, we will elucidate these relations by enumerating all possible cases of splitting and merging. With assumptions, we demonstrate the difficulty of involving edge relations provided that we have vertex splitting/merging.

We assume that all graphs for comparison are undirected and simple, i.e., they do not contain self-loop or multi-edges. Given two attributed graphs $G = (V_G, E_G, \mu_G, \xi_G)$ and $H = (V_H, E_H, \mu_H, \xi_H)$, we introduce the following notations:

- $E_{u,S_{mer}} = \{e_{ut}, t \in S_{mer}, e_{ut} \in E_G\}$ is set of edges connecting $u \in V_G$ to vertices $t \in S_{mer}$
- $E_{v,S_{spl}} = \{e_{vt}, t \in S_{spl}, e_{vt} \in E_H\}$ is set of edges connecting $v \in V_H$ to vertices $t \in S_{spl}$
- $E_{S^1_{mer},S^2_{mer}} = \{(u_i, u_j) \in E_G | u_i \in S^1_{mer}, u_j \in S^2_{mer}\}$ is the set of edges formed by two different sets of merged vertices, $S^1_{mer} \cap S^2_{mer} = \emptyset$.
- $p = |E_{S^1_{mer},S^2_{mer}}|$ is the cardinality of $E_{S^1_{mer},S^2_{mer}}$

- k_{mer} is the maximum split vertex, i.e., $\{u_i\} \to v, |\{u_i\}| \leq k_{mer}$, $k_{mer} \leq k$
- k_{spl} is the maximum merged vertex, i.e., $u \to \{v_i\}, |\{v_i\}| \leq k_{spl}$, $k_{spl} \leq k$.

Only one splitting case We assume the most simple scenario that one vertex $u \in V_G$ is split into maximum two vertices in H, i.e., $k_{spl} = 2$. A demonstration is in Fig. 4.3. Tab. 4.2 presents all possible edge operations in this case. So, *vertex might result in edge splitting* in addition to classical edge operations (substitution, deletion, and insertion).

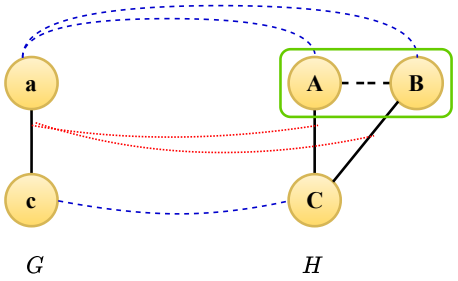


Figure 4.3: Example of edge relations induced when having vertex splitting $a \to \{A, B\}$ and vertex substitution $c \to C$: deletion of edge e_{AB} (in black dashed line), and splitting edge $e_{AC} \to \{e_{AC}, e_{BC}\}$ (in red dotted lines).

Status in E_G	Status in E_H	Operation	Notation
$\exists e_{ac}$	$\nexists e_{AC} \wedge \nexists e_{BC}$	deletion	$e_{ac} \to \varepsilon$
	$\exists e_{AC} \vee \exists e_{BC}$	substitution	$e_{ac} \to e_{AC}/e_{BC}$
	$\exists e_{AC} \wedge \exists e_{BC}$	**splitting**	$e_{ac} \to \{e_{AC}, e_{BC}\}$
$\nexists e_{ac}$	$\exists e_{AC} \wedge \exists e_{BC}$	insertion	$\varepsilon \to e_{AC}/e_{BC}$

Table 4.2: With vertex splitting $a \to \{A, B\}$ and substitution $c \to C$ simultaneously, possible edge relations can be induced on adjacent edges of these vertices.

Only one merging case Here we consider the merging of vertices where a set of vertices in G are merged into one vertex in H with $k_{mer} = 2$. The possibilities are shown in Table 4.3. Similarly, *vertex merging might lead to edge merging* as in the case of vertex splitting previously.

Splitting and merging with $k_{mer} = k_{spl} = k$ Now, we consider the more complicated case where there are vertex splitting and vertex merging at the same time. Firstly, we suppose that $k_{mer} = k_{spl} = k$ as in Fig. 4.5. All possible edge relations are enumerated in Table 4.4. We observe that *it is totally possible to have edge merging and splitting* in this case. Besides, we also have *substitution of edge sets*. This operation is therefore still called substitution or not? It is perhaps named Set-Substitution, and then a specific cost function should be given for this new operation. Besides, the set-substitution of egdes can be considered as many-to-many mapping of edges, which shows that the multivalent

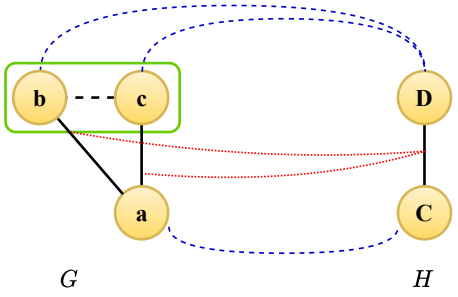


Figure 4.4: Example of edge relations induced when having vertex merging $\{b, c\} \to D$ and vertex substitution $a \to C$: deletion of edge e_{bc} (in black dashed line), and merging edge $\{e_{ba}, e_{ca}\} \to e_{DC}$ (in red dotted lines).

Status in E_G	Status in E_H	Operation	Notation
$\exists e_{ab} \wedge \exists e_{ac}$ $\exists e_{ab} \vee \exists e_{ac}$	$\nexists e_{DC}$	deletion	$e_{ab}/e_{ac} \to \varepsilon$
$\exists e_{ab} \wedge \exists e_{ac}$	$\exists e_{DC}$	**merging**	$\{e_{ab}, e_{ac}\} \to e_{DC}$
$\nexists e_{ab} \wedge \nexists e_{ac}$	$\exists e_{DC}$	insertion	$\varepsilon \to e_{DC}$
$\exists e_{ab} \vee \exists e_{ac}$	$\exists e_{DC}$	substitution	$e_{ab}/e_{ac} \to e_{DC}$

Table 4.3: With vertex splitting $a \to \{A, B\}$ and substitution $c \to C$ simultaneously, possible edge relations can be induced on adjacent edges of these vertices.

GM can result in many-to-many matching. In addition to that, we can observe there is deletion/insertion of set of edges. Nevertheless, these operations can be decomposed into two deletions/insertions of edges, which equivalent to edge deletion/insertion in classical GED.

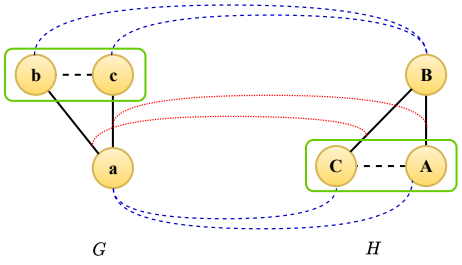


Figure 4.5: Example of edge relations induced when having vertex splitting $a \to \{C, A\}$ and merging $\{b, c\} \to B$ at the same time with $k_{mer} = k_{spl}$: deletion of edges e_{bc}, e_{CA} (dashed black lines), substitutions : $e_{ba} \to e_{BC}, e_{ca} \to e_{CA}$ (dotted red lines).

Splitting and merging with $k_{spl} \neq k_{mer}$ In this part, we assume that the maximum number of splitting/merging vertices are different, as shown in Fig. 4.6. So, it will be more complicated than the previous case. Possibilities of edge operations are listed in Table 4.5. *There are more possibilities for edge operations.* Also, *there is the substitution of edge sets with different sizes.* So, one question is: it should be still a substitution or a splitting?

Status of edges in E_G	Status of edges in E_H	Operation	Notation
$\exists e_{ab} \wedge \exists e_{ac}$	$\exists e_{BA} \wedge \exists e_{BC}$	set-substitution	$\{e_{ab}, e_{ac}\} \to \{e_{BA}, e_{BC}\}$
	$\exists e_{BA} \vee \exists e_{BC}$	**merging**	$\{e_{ab}, e_{ac}\} \to e_{BA}/e_{BC}$
	$\nexists e_{BA} \wedge \nexists e_{BC}$	deletion	$e_{ab} \to \varepsilon$, $e_{ac} \to \varepsilon$
$\exists e_{ab} \vee \exists e_{ac}$	$\exists e_{BA} \wedge \exists e_{BC}$	**splitting**	$e_{ab}/e_{ac} \to \{e_{BA}, e_{BC}\}$
	$\exists e_{BA} \vee \exists e_{BC}$	substitution	$e_{ab}/e_{ac} \to e_{BA}/e_{BC}$
	$\nexists e_{BA} \wedge \nexists e_{BC}$	deletion	$e_{ab}/e_{ac} \to \varepsilon$
$\nexists e_{ab} \vee \nexists e_{ac}$	$\exists e_{BA} \wedge \exists e_{BC}$ $\exists e_{BA} \vee \exists e_{BC}$	insertion	$\varepsilon \to e_{BA}, \varepsilon \to e_{BC}$

Table 4.4: With vertex splitting $a \to \{C, A\}$ and merging $\{b, c\} \to B$ simultaneously, possible edge relations can be induced on adjacent edges of these vertices. It depends on the existence of these edges.

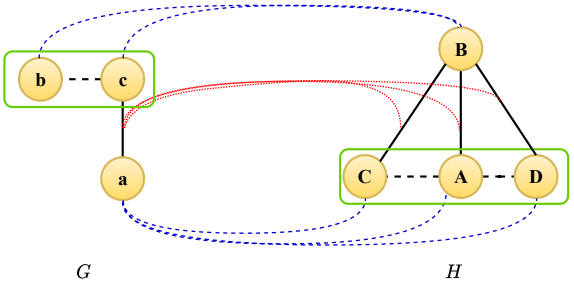


Figure 4.6: Example of edge relations induced when having vertex splitting $a \to \{C, A, D\}$ and merging $\{b, c\} \to B$ simultaneously with $k_{mer} \neq k_{spl}$: edge deletions for e_{bc}, e_{CA}, e_{AD} (dashed black lines), edge merging for $e_{ba} \to \{e_{BC}, e_{BA}, e_{BD}\}$ (dotted red lines).

Status in E_G	Status in E_H	Operation	Notation
$\exists e_{ab} \wedge \exists e_{ac}$	$\exists e_{BA} \wedge \exists e_{BC} \wedge \exists e_{BD}$	set-substitution	?
	$\exists e_{BA} \vee \exists e_{BC} \vee \exists e_{BD}$	**merging**	$\{e_{ab}, e_{ac}\} \to e_{BA}/e_{BC}/e_{BD}$
	$\exists e_{BA} \wedge \exists e_{BC} \wedge \nexists e_{BD}$	set-substitution	$\{e_{ab}, e_{ac}\} \to \{e_{BA}, e_{BC}\}$
	$\nexists e_{BA} \wedge \nexists e_{BC} \wedge \nexists e_{BD}$	deletion	$e_{ab} \to \varepsilon, e_{ac} \to \varepsilon$
$\nexists e_{ab} \wedge \nexists e_{ac}$	$\lvert\{e_j\}\rvert \geq 1$	insertion	$\varepsilon \to \{e_j\}$
$\exists e_{ab} \vee \exists e_{ac}$	$\lvert\{e_j\}\rvert \in [2, 3]$	**splitting**	$e_{ab} \to \{e_j\}$
	$\lvert\{e_j\}\rvert = 1$	substitution	$e_{ab} \to e_j$
	$\lvert\{e_j\}\rvert = 0$	deletion	$e_{ab} \to \varepsilon$

Table 4.5: Possible edge operations on adjacent edges of vertex splitting and vertex merging at the same time with $k_{spl} \neq k_{mer}$.

From cases of vertex splitting and vertex merging happening at the same time, we summarize all possibilities in Tab. 4.6. We observe that the edge operations rely on the cardinalities of $E_{u,S_{mer}}$ and $E_{v,S_{spl}}$, $u \in V_G, v \in V_H$ as shown in Tab. 4.5. Nevertheless, we let an open question in case of two sets of edges in two graphs matching together, i.e., $\big((E_{u,S_{mer}} \geq 1) \wedge (E_{v,S_{spl}} \geq 1)\big)$.

Case	Operation	Notation
$\lvert E_{u,S_{mer}}\rvert = 0 \wedge \lvert E_{v,S_{spl}}\rvert \neq 0$	insertion	$\varepsilon \rightarrow E_{v,S_{spl}}$
$\lvert E_{u,S_{mer}}\rvert \neq 0 \wedge E_{v,S_{spl}} = 0$	deletion	$E_{u,S_{mer}} \rightarrow \varepsilon$
$\lvert E_{u,S_{mer}}\rvert = 1 \wedge E_{v,S_{spl}} = 1$	substitution	$E_{u,S_{mer}} \rightarrow E_{v,S_{spl}}$
$\lvert E_{u,S_{mer}}\rvert = 1 \wedge E_{v,S_{spl}} \geq 1$	split	$E_{u,S_{mer}} \rightarrow E_{v,S_{spl}}$
$\lvert E_{u,S_{mer}}\rvert \geq 1 \wedge E_{v,S_{spl}} = 1$	merging	$E_{u,S_{mer}} \rightarrow E_{v,S_{spl}}$
$\lvert E_{u,S_{mer}}\rvert \geq 1 \wedge E_{v,S_{spl}} \geq 1$	**substitution ?**	$E_{u,S_{mer}} \rightarrow E_{v,S_{spl}}$

Table 4.6: Generality of possible edge operations between two graphs induced by vertex splitting and vertex merging at the same times.

Two merging cases Now, we consider the case where two sets of vertices in G are merged to two different vertices in H, i.e., $S^1_{mer} \rightarrow v_k, S^2_{mer} \rightarrow v_l$, $v_k, v_l \in V_H, v_k \neq v_l$. An illustration is in Fig. 4.7. All possible cases are listed in Tab. 4.7. We see that *there is edge merging when having two sets of vertices mapped to two separate vertices in other graph.*

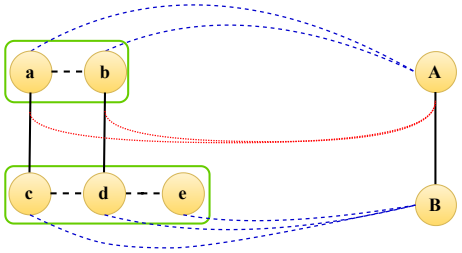


Figure 4.7: Example of edge relations induced by vertex merging $\{a, b\} \rightarrow A$ and $\{c, d, e\} \rightarrow B$ simultaneously: edge deletions for e_{ab}, e_{cd}, e_{de} (dashed black lines), edge merging for $\{e_{ac}, e_{bd}\} \rightarrow e_{AB}$ (dotted red lines).

Case	Operation	Notation
$\lvert E_{u,S_{mer}}\rvert = 0 \wedge \exists e_{v_k v_l}$	insertion	$\varepsilon \rightarrow e_{v_k v_l}$
$\lvert E_{u,S_{mer}}\rvert \geq 1 \wedge \nexists e_{v_k v_l}$	deletion	$E_{S^1_{mer},S^2_{mer}} \rightarrow \varepsilon$
$\lvert E_{u,S_{mer}}\rvert = 1 \wedge \exists e_{v_k v_l}$	substitution	$E_{S^1_{mer},S^2_{mer}} \rightarrow e_{v_k v_l}$
$\lvert E_{u,S_{mer}}\rvert \geq 2 \wedge \exists e_{v_k v_l}$	**merging**	$E_{S^1_{mer},S^2_{mer}} \rightarrow e_{v_k v_l}$

Table 4.7: Possible edge operations when two separate vertex sets in G are merged to two different vertices in H, where $e_{v_k v_l} \in E_H$.

Two splitting cases It is done as for two merging cases but in an inverse direction. One example is given in Fig. 4.8. Theoretically, we can have edge splitting in this case. All the possibilities are presented in Table 4.8.

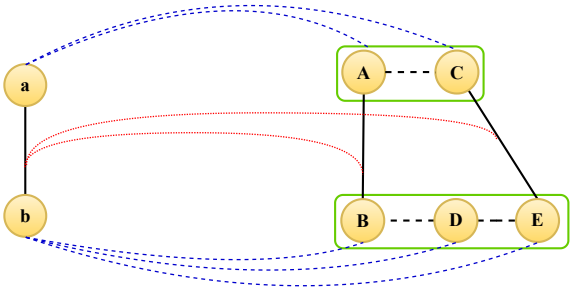


Figure 4.8: Example of edge relations induced by vertex splitting $a \to \{A, C\}$ and $B \to \{B, D, E\}$ simultaneously: edge deletions for e_{AC}, e_{BD}, e_{DE} (dashed black lines), edge splitting for $e_{ab} \to \{e_{AB}, e_{CE}\}$ (dotted red lines).

Case	Operation	Notation
$\nexists e_{u_i u_j} \wedge \lvert E_{u,S_{mer}} \rvert \geq 1$	insertion	$\varepsilon \to E_{S^1_{spl}, S^2_{spl}}$
$\exists e_{u_i u_j} \wedge \lvert E_{u,S_{mer}} \rvert = 0$	deletion	$e_{u_i u_j} \to \varepsilon$
$\exists e_{u_i u_j} \wedge \lvert E_{u,S_{mer}} \rvert = 1$	substitution	$e_{u_i u_j} \to E_{S^1_{spl}, S^2_{spl}}$
$\exists e_{u_i u_j} \wedge \lvert E_{u,S_{mer}} \rvert \geq 2$	**splitting**	$e_{u_i u_j} \to E_{S^1_{spl}, S^2_{spl}}$

Table 4.8: Possible edge operations when two separate vertices in G are split to two different vertex sets in H, where $e_{u_i u_j} \in E_G$.

4.2.2.3 Corresponding cost functions

In subsection 4.2.2.2, we see the complication of defining the edge relations when having vertex splitting/merging. This task requires the verification of the existence of edges in both graphs to decide which operation to apply afterward. In addition, once a specific operation is applied, it is essential to determine a cost corresponding to that operation. This step is already complicated in GED, it is even harder for ExGED due to much more possibilities. Furthermore, it will be more challenging when the value of k increases. That means more evaluations need to be done to exactly consider the edge operations in vertex splitting/merging. So, it is crucial that the edge oprerations will be exactly computed? It depends on the specific application and the user's target. However, one should spend more computational effort to find the exact solution for ExGED.

We give a general way to define the cost for each operation applied on vertices and edges. Given two attributed graphs, $G = (V_G, E_G, \mu_G, \xi_G)$ and $H = (V_H, E_H, \mu_H, \xi_H), u \in V_G, v \in V_H, e_{ij} \in E_G, e_{kl} \in E_H$, we have all the possible edit operations for ExGED as follows:

1. Substitution
   - Vertex
   $$c(u \to v) = \Upsilon\left(\mu_G(u), \mu_H(v)\right)$$
   - Edge
   $$c(e_{ij} \to e_{kl}) = \Upsilon\left(\xi_G(e_{ij}), \xi_H(e_{kl})\right)$$

- Edge set

$$c(\{e_{ij}\} \rightarrow \{e_{kl}\}) = \Upsilon\left(\xi_G(\{e_{ij}\}), \xi_H(\{e_{kl}\})\right)$$

2. Deletion

- Vertex:

$$c(u \rightarrow \varepsilon) = \Upsilon\left(\mu_G(u)\right)$$

- Edge:

$$c(e_{ij} \rightarrow \varepsilon) = \Upsilon\left(\xi_G(e_{ij})\right)$$

3. Insertion

- Vertex

$$c(\varepsilon \rightarrow v) = \Upsilon\left(\mu_H(v)\right)$$

- Edge

$$c(\varepsilon \rightarrow e_{kl}) = \Upsilon\left(\xi_H(e_{kl})\right)$$

4. Merging

- Vertex

$$c(S_{mer} \rightarrow v) = \Upsilon\left(\mu_G(u), \mu_H(v)\right), \forall u \in S_{mer}$$

- Edge

$$c(S^e_{mer} \rightarrow e_{kl}) = \Upsilon\left(\xi_G(e_{ij}), \xi_H(e_{kl})\right), \forall e_{ij} \in S^e_{mer}$$

5. Splitting

- Vertex

$$c(u \rightarrow S_{spl}) = \Upsilon\left(\mu_G(u), \mu_H(v)\right), \forall v \in S_{spl}$$

- Edge

$$c(e_{ij} \rightarrow S^e_{spl}) = \Upsilon\left(\xi_G(e_{ij}), \xi_H(e_{kl})\right), \forall e_{kl} \in S^e_{spl}$$

6. 'Looping edge' (E_{loop})

$$c(e) = \Upsilon\left(\xi(e)\right), \forall e \in E_{loop}$$

where Υ is a function that needs to be defined in advance and it can depend on the vertex/edge attributes as well as the specific problem; $S^e_{mer}(S^e_{spl})$ is the set of merging (splitting) edges induced by vertex merging (splitting); the 'looping edge' is the edges between the merged (split) vertices.

In this thesis, the costs of edge operations involved by vertex splitting and merging are considered approximately. Because of the high complexity of ExGED, especially the edge relations in the extended cases, it is not trivial to take into account all of them during the calculation process. We relax some complicated cases such as substitutions of two sets of edges but we still keep in mind about that in order not to loose the generality. Precisely, the cost matrix is designed for estimating the goodness of each vertex mapping from two graphs. Therefore, it is just the first-order, not the second-order. As consequence, we do not consider two couples of vertices at the same time (second-order), and the sets of edges will not happen in this case. In the next section, we explain how we propose to consider them with some appropximations in order to ensure the trade-off between heavy computational process and taking into account the structural properties of the graphs locally.

4.2.3 Enrich cost matrix with estimated costs for edge operations

In the previous section, we analyze edge relations when having vertex merging/splitting. It is very complicated to define exactly these relations. So, it will not be practical to solve ExGED as QAP exactly. Following the work of Riesen et al. [2007b], we integrate the estimated costs of edge operations for each vertex operation. Consequently, the cost matrix $\mathbf{C}$ of ExGED is enriched with the local topology of each vertex. For ExGED, we have more cases than traditional GED with splitting and merging. As a result, we consider also the split/merged edges during the approximation based on the analysis done in the subsection 4.2.2.2. By that means, we can obtain more heuristic information, which will be more advantageous for the solver afterward.

The approximation of costs for vertex substitution, deletion, and insertion is similar to that in Riesen et al. [2007b]. For vertex merging and splitting, we introduce one set called E_{loop} to consider edges between split/merged vertices. Moreover, cost matrix for edges is also built as for vertex substitution. Given two attributed graphs $G = (V_G, E_G, \mu_G, \xi_G)$ and $H = (V_H, E_H, \mu_H, \xi_H)$, details of computation are as follows:

- Vertex substitution $u_i \rightarrow v_j$, $(u_i \in V_G, v_j \in V_H)$: let E_{u_i}, E_{v_j} be sets of incident edges of u_j and v_j, respectively. Then, a square edge cost matrix $\mathbf{Ce}$ is built from E_{u_i}, E_{v_j} with respect to the edge cost functions defined in advance. The size of $\mathbf{Ce}$ is $(|E_{u_i}| + |E_{v_j}|) \times (|E_{u_i}| + |E_{v_j}|)$. Next, we apply the Munkres algorithm (more details in the Appendix B) to find the minimum sum of edge operation costs Munkres [1957], or

$$c_{i,j} = c(u_i \rightarrow v_j) + Munkres(\mathbf{Ce})$$

- Vertex deletion $u_i \rightarrow \varepsilon (u_i \in V_G)$: deleting a vertex will remove all its adjacent edges, or

$$c_{i,\varepsilon} = c(u_i \rightarrow \varepsilon) + \sum_{e \in E_{u_i}} c(e \rightarrow \varepsilon)$$

- Vertex insertion $\varepsilon \to v_j (v_j \in V_H)$: inserting a vertex will insert all its adjacent edges, or

$$c_{\varepsilon,j} = c(\varepsilon \to v_j) + \sum_{e \in E_{v_j}} c(\varepsilon \to e)$$

- Vertex merging $S_{mer} \to v_j$:
  - We compute first the sets of incident edges to vertices in S_{mer} and v, denoted $E_{S_{mer}}$ and E_{v_j}, respectively.
  - Let E_{loop} be the set of edges connecting the vertices $u_i \in S_{mer}$, we have $E'_{S_{mer}} = E_{S_{mer}} \setminus E_{loop}$ is the set of incident edges without 'looping edges' in S_{mer}.
  - After, an edge cost matrix is also constructed for two sets $E'_{S_{mer}}$ and E_{v_j} similarly to vertex substitution.
  - We also utilize Munkres algorithm to find the minimal cost for $E'_{S_{mer}}$ and E_{v_j}. Finally, the cost for vertex merging is as follows:

$$c_{S_{mer},j} = c(S_{mer} \to v_j) + Munkres(\mathbf{Ce}) + \sum_{e \in E_{loop}} c(e \to \varepsilon)$$

- Vertex splitting $u_i \to S_{spl}$: we compute similarly to vertex merging.
  - Firstly, we construct the sets of incident edges of u_i and S_{spl}, denoted E_{u_i} and $E_{S_{spl}}$, respectively.
  - Secondly, we find the set of 'looping edges' in S_{spl}. Then, we get the set $E'_{S_{spl}} = E_{S_{spl}} \setminus E_{loop}$ without 'looping edges'
  - Thirdly, an edge cost matrix $\mathbf{Ce}$ is established for E_{u_i} and $E'_{S_{spl}}$, and Munkres algorithm is used to search for a minimum edge cost.
  - Finally, the cost of vertex splitting will be computed as:

$$c_{u_i,S_{spl}} = c(u_i \to S_{spl}) + Munkres(\mathbf{Ce}) + \sum_{e \in E_{loop}} c(\varepsilon \to e)$$

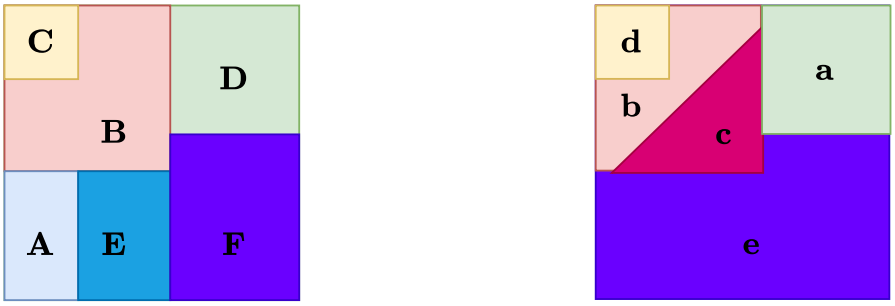
4.3 Illustrative example

In the preceding section, we have presented how to build, simplify, and compute the cost matrix for ExGED in detail. Now, we will demonstrate an example of these steps. In the example, the main steps are as follows:

- introducing the problem;
- doing graph representation;
- determining candidates for splitting and merging;

- defining cost functions for edit operations on vertices and edges;
- building the cost matrix $\mathbf{C}$ for vertices;
- computing values for each block in $\mathbf{C}$;
- updating the costs to consider also costs of edge operations.

Assuming that we have two segmented images, one is the ground truth (Fig. 4.9i), and another is the result obtained from a segmentation algorithm (Fig. 4.9ii). In Fig. 4.9ii, there are both over-segmentation and under-segmentation. So, in this case, the multivalent GM would be more useful than univalent GM. From the segmented images, we build the graph representation, the vertices are segments, and the edges are the 2D relations between these segments. We suppose that the attributes of vertices are the area of their corresponding segments (ϖ_{v_i}), and there is no attribute for edges. All attributes are normalized. Fig. 4.10 demonstrates graph representations of two segmented images.



(i) (ii)

Figure 4.9: Segmentation result from an image: i) ground truth, ii) result from an algorithm.

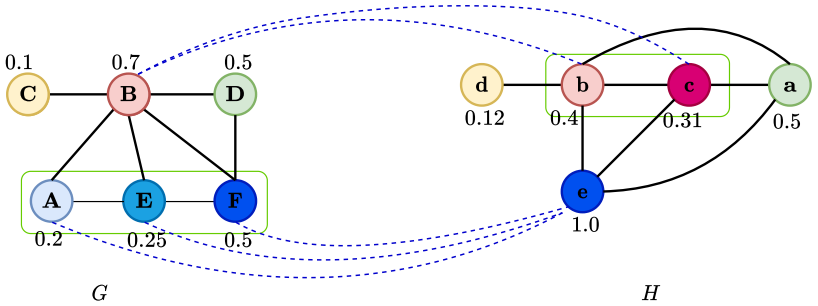


Figure 4.10: Graph representations G and H of two segmentations in Fig. 4.9i and Fig. 4.9ii, respectively; a partial edit path $\lambda^{partial} = \Big\{\big(B, \{b, c\}\big), \big(\{A, E, F\}, e\big)\Big\}$ is demonstrated with dashed blue lines between G and H.

From the graph representation step, $G = (V_G, E_G, \mu_G, \xi_G)$ is ground truth graph (target graph), and $H = (V_H, E_H, \mu_H, \xi_H)$ is a sample graph (source graph). The purpose is to find the correspondence between the target graph and the source graph. It is a GM problem,

and we use multivalent GM for this task. We have $m = |V_G| = 6$, $n = |V_H| = 5$. We suppose that one vertex is mapped to a maximum of 3 other vertices, i.e., $k = 3$. If all connected vertices can be candidates for splitting and merging, all possibilities for merging and splitting sets with $k = 3$ will be:

- $|P^k_{mer}| = \binom{|E_G|}{2} + \binom{|E_G|}{3} = 84$,
- $|P^k_{spl}| = \binom{|E_H|}{2} + \binom{|E_H|}{3} = 35$.

These possibilities are much larger than the graph's initial orders. To demonstrate some of them, we assume that the candidates for merging and splitting of vertices are only:

$$\begin{aligned} P^k_{mer} = &\quad \{S_{mer}\} = \Big\{\{A,E\},\{E,F\},\{A,E,F\},\{B,C\},\{B,D\}\Big\} \Rightarrow h = 5 \\ P^k_{spl} = &\quad \{S_{spl}\} = \Big\{\{a,c\},\{b,c\},\{b,d\},\{c,e\}\Big\} \Rightarrow l = 4 \end{aligned}$$

Then, the extended sets of vertices in G and H are as follows.

$$\begin{aligned} V^k_G &= V_G \cup \{\varepsilon\} \cup P^k_{mer} \\ &= \Big\{A,B,C,D,E,F,\varepsilon,\{A,E\},\{E,F\},\{A,E,F\},\{B,C\},\{B,D\}\Big\} \\ V^k_H &= V_H \cup \{\varepsilon\} \cup P^k_{spl} \\ &= \Big\{a,b,c,d,e,\varepsilon,\{a,c\},\{b,c\},\{b,d\},\{c,e\}\Big\} \end{aligned}$$

The cost functions of vertex operations are based on the area of the segment ϖ_{v_i}. The costs of edge operations are defined in $[0,1]$. Details are in Tab. 4.9. Due to problem context, both vertices in two graphs must match together. Therefore, we set high costs to deletion and insertion of vertices. There is no label on edges, so the cost of edge substitution is 0. The costs for deleting/inserting 'looping edge' induced by vertex merging/splitting will be less than the cost of deleting/inserting edges. Alslo, the cost functions favor substitution over other operations.

$$\mathbf{C} = \left(\begin{array}{ccc|c|cccc} \boldsymbol{c_{A,a}} & \cdots & c_{A,e} & \boldsymbol{c_{A,\varepsilon}} & c_{A,\{a,c\}} & c_{A,\{b,c\}} & c_{A,\{b,d\}} & c_{A,\{c,e\}} \\ c_{B,a} & \cdots & c_{B,e} & c_{B,\varepsilon} & c_{B,\{a,c\}} & \boldsymbol{c_{B,\{b,c\}}} & c_{B,\{b,d\}} & c_{B,\{c,e\}} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{F,a} & \cdots & c_{F,e} & c_{F,\varepsilon} & c_{F,\{a,c\}} & c_{F,\{b,c\}} & c_{F,\{b,d\}} & c_{F,\{c,e\}} \\ \hline \boldsymbol{c_{\varepsilon,a}} & \cdots & c_{\varepsilon,e} & 0 & \multicolumn{4}{c}{\infty} \\ \hline c_{\{A,E\},a} & \cdots & c_{\{A,E\},e} & & & & & \\ c_{\{E,F\},a} & \cdots & c_{\{E,F\},e} & & & & & \\ c_{\{A,E,F\},a} & \cdots & \boldsymbol{c_{\{A,E,F\},e}} & \infty & \multicolumn{4}{c}{\infty} \\ c_{\{B,C\},a} & \cdots & c_{\{B,C\},e} & & & & & \\ c_{\{B,D\},a} & \cdots & c_{\{B,D\},e} & & & & & \end{array}\right) \tag{4.3}$$

The size of the cost matrix 4.3 is $(m+1+h)\times(n+1+l) = 12\times 10$. We will not present in detail the computational detail for each position in $\mathbf{C}$. Nevertheless, we show below how to compute cost for some types of edit operations. They are vertex substitution's cost ($c_{A,a}$), vertex deletion's cost ($c_{A,\varepsilon}$), vertex insertion's cost ($c_{\varepsilon,a}$), vertex splitting's cost ($c_{B,\{b,c\}}$).

Operation	Vertex operation cost	Edge operation cost
Substitution	$c(u \to v) = \lvert\varpi_u - \varpi_v\rvert$	$c(e_G \to e_H) = 0$
Deletion	$c(u \to \varepsilon) = 1$	$c(e_G \to \varepsilon) = 0.5$
Insertion	$c(\varepsilon \to v) = 1$	$c(\varepsilon \to e_H) = 0.5$
Merging	$c(S_{mer} \to v) = \left\lvert \left(\sum_{u_i \in S_{mer}} u_i \right) - \varpi_v \right\rvert$	$c(e \to \varepsilon) = 0.1, e \in E_{loop}$
Splitting	$c(u \to S_{spl}) = \left\lvert \varpi_u - \left(\sum_{v_j \in S_{spl}} v_j \right) \right\rvert$	$c(\varepsilon \to e) = 0.1, e \in E_{loop}$

Table 4.9: Cost functions of edit operations to compare two graphs in Fig. 4.10.

1. Vertex substitution: $c_{A,a} = c(A \to a) + Munkres(\mathbf{Ce})$, where
   - $c(A \to a) = |\varpi_A - \varpi_a| = |0.2 - 0.5| = 0.3$
   - $E_A = \{AB, AE\}, E_a = \{ab, ac, ae\}$, then $Munkres(\mathbf{Ce}) = 0.5$

$$\mathbf{Ce} = \begin{array}{c|ccc|cc} & ab & ac & ae & \varepsilon_{AB} & \varepsilon_{AE} \\ \hline AB & c_{AB,ab} & c_{AB,ac} & c_{AB,ae} & c_{AB,\varepsilon} & \infty \\ AE & c_{AE,ab} & c_{AE,ac} & c_{AE,ae} & \infty & c_{AE,\varepsilon} \\ \hline \varepsilon_{ab} & c_{\varepsilon,ab} & & \infty & & \\ \varepsilon_{ac} & & c_{\varepsilon,ac} & & \mathbf{0} & \\ \varepsilon_{ae} & \infty & & c_{\varepsilon,ae} & & \end{array}$$

$$= \begin{array}{c|ccc|cc} & ab & ac & ae & \varepsilon_{AB} & \varepsilon_{AE} \\ \hline AB & 0 & 0 & 0 & 0.5 & \infty \\ AE & 0 & 0 & 0 & \infty & 0.5 \\ \hline \varepsilon_{ab} & 0.5 & & \infty & & \\ \varepsilon_{ac} & & 0.5 & & \mathbf{0} & \\ \varepsilon_{ae} & \infty & & 0.5 & & \end{array}$$

   - So, $c_{A,a} = 0.3 + 0.5 = 0.8$
2. Vertex deletion $c_{A,\varepsilon} = 1 + 2 * 0.5 = 2$
3. Vertex insertion $c_{\varepsilon,a} = 1 + 3 * 0.5 = 2.5$
4. Vertex merging $c_{\{A,E,F\},e} = |(\varpi_A + \varpi_E + \varpi_F) - \varpi_a| + Munkres(\mathbf{Ce}) + \sum_{e \in E_{loop}} c(e \to \varepsilon)$, where
   - $E_{\{A,E,F\}} = \{AB, AE, EB, EF, FB, FD\}$
   - $E_{loop} = \{AE, EF\}$

- Then, $E'_{\{A,E,F\}} = E_{\{A,E,F\}} \setminus E_{loop} = \{AB, EB, FB, FD\}$
- $E_e = \{ea, eb, ec\}$
- $Munkres(\mathbf{Ce}) = 0 + 0.5 = 0.5$

$$\mathbf{Ce} = \left(\begin{array}{cccc|cccc} c_{AB,ea} & c_{AB,eb} & c_{AB,ec} & c_{AB,\varepsilon} & & & & \infty \\ c_{EB,ea} & c_{EB,eb} & c_{EB,ec} & & c_{EB,\varepsilon} & & & \\ c_{FB,ea} & c_{FB,eb} & c_{FB,ec} & & & c_{FB,\varepsilon} & & \\ c_{FD,ea} & c_{FD,eb} & c_{FD,ec} & \infty & & & & c_{FD,\varepsilon} \\ \hline c_{\varepsilon_{ea}},ea & & \infty & & & & & \\ & c_{\varepsilon_{eb}},eb & & & \mathbf{0} & & & \\ \infty & & c_{\varepsilon_{ec}},ec & & & & & \end{array}\right)$$

$$= \left(\begin{array}{ccc|cccc} 0 & 0 & 0 & 0.5 & & & \infty \\ 0 & 0 & 0 & & 0.5 & & \\ 0 & 0 & 0 & & & 0.5 & \\ 0 & 0 & 0 & \infty & & & 0.5 \\ \hline 0.5 & & \infty & & & & \\ & 0.5 & & & \mathbf{0} & & \\ \infty & & 0.5 & & & & \end{array}\right)$$

- $c_{\{A,E,F\},e} = |(0.2 + 0.25 + 0.5) - 1| + 0.5 + 0.1 * 2 = 0.75$

5. Vertex splitting $c_{B\to\{b,c\}} = |\varpi_B - (\varpi_b + \varpi_c)| + Munkres(\mathbf{Ce}) + \sum_{e\in E_{loop}} c(e \to \varepsilon)$, where

- $E_B = \{BA, BC, BD, BE, BF\}$
- $E_{\{b,c\}} = \{bc, bd, be, ca, ce\}$
- $E_{loop} = \{bc\}$
- Then, $E'_{\{b,c\}} = E_{\{b,c\}} \setminus E_{loop} = \{bd, be, ca, ce\}$
- $Munkres(\mathbf{Ce}) = 0 * 4 + 0.5 = 0.5$

$$\mathbf{Ce} = \left(\begin{array}{cccc|ccccc} c_{BA,bd} & c_{BA,be} & c_{BA,ca} & c_{BA,ce} & c_{BA,\varepsilon} & & & \infty & \\ c_{BC,bd} & c_{BC,be} & c_{BC,ca} & c_{BC,ce} & & c_{BC,\varepsilon} & & & \\ c_{BD,bd} & c_{BD,be} & c_{BD,ca} & c_{BD,ce} & & & c_{BD,\varepsilon} & & \\ c_{BE,bd} & c_{BE,be} & c_{BE,ca} & c_{BE,ce} & & & & c_{BE,\varepsilon} & \\ c_{BF,bd} & c_{BF,be} & c_{BF,ca} & c_{BF,ce} & \infty & & & & c_{FD,\varepsilon} \\ \hline c_{\varepsilon_{bd},bd} & & & \infty & \multicolumn{4}{c}{\multirow{4}{*}{$\mathbf{0}$}} \\ & c_{\varepsilon_{be},be} & & & & & & & \\ & & c_{\varepsilon_{ca},ca} & & & & & & \\ \infty & & & c_{\varepsilon_{ce},ce} & & & & & \end{array}\right)$$

$$= \left(\begin{array}{cccc|ccccc} 0 & 0 & 0 & 0 & 0.5 & & & \infty & \\ 0 & 0 & 0 & 0 & & 0.5 & & & \\ 0 & 0 & 0 & 0 & & & 0.5 & & \\ 0 & 0 & 0 & 0 & & & & 0.5 & \\ 0 & 0 & 0 & 0 & \infty & & & & 0.5 \\ \hline 0.5 & & & \infty & \multicolumn{4}{c}{\multirow{4}{*}{$\mathbf{0}$}} \\ & 0.5 & & & & & & & \\ & & 0.5 & & & & & & \\ \infty & & & 0.5 & & & & & \end{array}\right)$$

- $c_{B\to\{b,c\}} = |0.7 - (0.4 + 0.31)| + 0.5 + 0.1 = 0.61$

We repeat the above steps until all values in $\mathbf{C}$ are filled. Then, we use them to compute the cost (distance) for edit paths. For example, the partial edit path $\lambda^{partial}(G,H) = \Big\{\big(B,\{b,c\}\big),\big(\{A,E,F\},e\big)\Big\}$ in Fig. 4.10 has the corresponding cost:

$$d_{\lambda^{partial}}(G,H) = c_{B,\{b,c\}} + c_{\{A,E,F\},e} = 0.61 + 0.75 = 1.36.$$

The above example shows the detailed steps to calculate each vertex matching in an edit path (partial or complete). The cost is a combination of three elements: 1) the vertex cost, 2) the estimated local edge cost, and 3) the cost related to extended cases (looping edges and split/merged edges). The 1st and 2nd costs are calculated similarly as in Riesen et al. [2007b], the 3rd one is computed regarding the extended operation and the number of vertices (edges) involved. The process reveals the complicated calculation in this case. Therefore, it is a challenge to design a good solver for the ExGED, which will be presented in the next chapter.

4.4 Summary

In this chapter, we have formulated the multivalent GM as an extension of GED, called Extended Graph Edit Distance (ExGED). Specifically, from the initial definition of GED, the supplement of two edit operations, splitting and merging, allows considering the one-to-many and many-to-one mappings in multivalent GM. Therefore, ExGED has more mapping possibilities than GED and can cover more situations.

One traditional methodology to deal with GED is to design a cost matrix that presents the goodness of mappings between vertices. Then, an algorithm can be applied to this cost matrix to find the optimal or suboptimal solution. Following this methodology, we first construct a cost matrix $\mathbf{C}$ for vertex mappings in ExGED. The cost matrix is not squared as in the classical GED. We also analyze the dimension of $\mathbf{C}$. We show that size of $\mathbf{C}$ is influenced by the parameter k-the maximum number of vertices for splitting/merging. When k increases, size of $\mathbf{C}$ goes exponentially. Therefore, k should be well considered when dealing with ExGED.

Besides considering the goodness of vertex mappings, we also take into account the edge relations, especially when having vertex splitting/merging. We have done a comprehensive analysis of the edge relations in the extended cases. It shows that the edge relations are far more complicated than the vertex relations because several types of edge mappings can happen. To tackle this issue, we simplify the edge relations by only looking at the local structure of the considered vertices. Thus, the cost matrix $\mathbf{C}$ contains not only the vertex costs but also the edge costs. At each position $c_{i,j} \in \mathbf{C}$, the cost is composed of two components: vertex cost and local estimated edge cost. As a consequence, $\mathbf{C}$ is now more powerful because it considers both the attributes and the local topology of the vertices. In addition, we give an illustrative example to show how to compute the cost matrix in detail. This cost matrix will be used for the algorithm presented in the next chapter.

Chapter 5

Solving ExGED with ACO

Contents

5.1 ACO's design for ExGED 105
5.2 Experimental setup 111
5.3 Comparison ACO with the greedy and random search for ExGED problem 132
5.4 Evaluation between multivalent and univalent GM on symbol dataset 136
5.5 Summary 140

Abstract

In chapter 4, we have proposed a problem formulation for multivalent GM based on GED, called ExGED. We have explained concepts related to ExGED, such as additional operations (merging and splitting) and edge relations induced by vertex merging and splitting. We have also built a cost matrix $\mathbf{C}$ for ExGED to store costs of all possible vertex mappings. Moreover, $\mathbf{C}$ is enriched with estimated edge costs for each vertex mapping. In this chapter, we concentrate on how to solve ExGED with ACO, that is to say how to design a general pipeline of ACO for solving ExGED, and how ACO can employ cost matrix $\mathbf{C}$ as a heuristic information to search for qualitative solutions. Besides, some strategies are applied to reduce the algorithm complexity and enhance the solution quality. After that, we conduct a comprehensive analysis of the proposed algorithm on several aspects. First, it is about the ACO's properties: parameter tuning, pheromone distribution, stability of produced solutions, performance between its variants. After, we evaluate the performance of ACO against other meta-heuristic algorithms to verify its efficiency. Finally, we would like to know whether or not multivalent GM is better than univalent GM and in which context.

5.1 ACO's design for ExGED

In the chapter 3, we have introduced ACO in a general way. It is a meta-heuristic method and belongs to the family of global-search algorithms. ACO is generic, so it is

easy to apply to different problems. We have provided a demonstration of ACO for TSP in section 3.2.2, which is also a combinatorial problem as the GM problem. Despite that, it needs some adaptations in each context. Therefore, we present here how to design ACO for solving the multivalent GM in the form of ExGED. Firstly, we show how to build a search space for the ant colony, called construction graph. The search space of ExGED is far more complicated than in the case of TSP due to much more candidates. Then, we propose ACO for tackling the ExGED. In the approach, we specify: how each ant builds a complete edit path, how to distribute the pheromone, and how to improve the solution over iterations. In a first attempt, a general approach is used with a complete search space. Then, a neighborhood strategy is applied to reduce the search space, which helps to speed up the calculation process. Finally, we summarize the main steps in a general pipeline.

5.1.1 Construction graph: a search space for ACO

In ACO, ants are agents which moves on a graph, called G_{ants}, in order to build a solution. The construction graph G_{ants} provides all possible vertex mappings (included extended cases) between two graphs G and H. In other words, G_{ants} is built to explore all the possible edit paths for the ExGED problem defined in the previous section. The candidates for each edit operation are the Cartesian products of vertex sets (including merging and splitting) as in Table 5.1. G_{ants} is complete and undirected. Fig. 5.1 presents the construction graph built from the graphs of Fig. 4.10. For edges that violate the constraints of ExGED such as the edge between (A, a) and (A, b) in Fig. 5.1, a very high cost is set to forbid ants moving between these candidates by checking their indicies. The pheromone values (τ) are laid on the vertices of G_{ants}. The pheromone amount reflects the learnt desirability of the ant colony. When a vertex in G_{ants} is visited several times, it will have a high amount of pheromone, indicating that this vertex contributes strongly to the good solution. This fact is true for most ACO variants, with the exception of ACS.

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

Table 5.1: Candidates for each edit operation in the construction graph, where V_G, V_H are vertex sets in G, H, respectively, ε is a virtual vertex, P^k_{mer}, P^k_{spl} are sets of merging and splitting vertices, respectively.

Remark that the possibility of an ant to move on G_{ants} depends on the current partial edit path. For instance, according to Fig. 4.10 and Fig. 5.1, the current partial edit path is $\lambda^{partial} = \Big\{\big(B, \{b, c\}\big), \big(\{A, E, F\}, e\big)\Big\}$ (red color vertices in Fig. 5.1). Therefore, all related candidates to these operations are pruned accordingly (in gray color). That means the current ant cannot move on these positions at the moment. The remaining candidates the ants can visit are in black. This action is to respect the constraint that each vertex is selected only one time in ExGED. It is similar to the TSP. One should distinguish the construction graph G_{ants} introduced here with the one in the work of [Sammoud et al., 2005]. There are purely vertex substitutions in that construction graph. As a result, its size is smaller and less complicated.

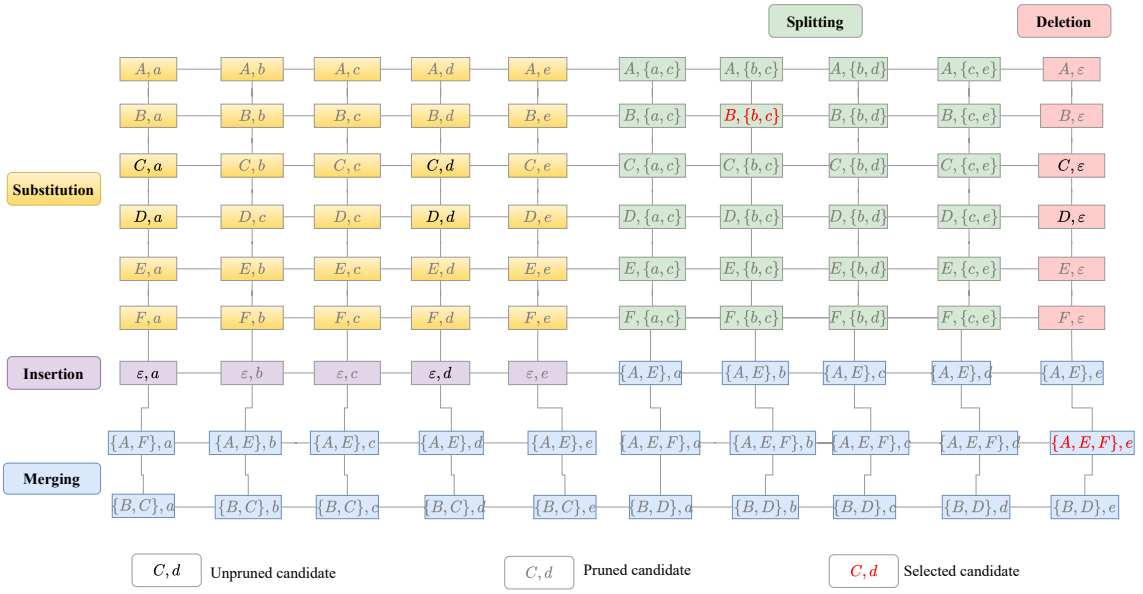


Figure 5.1: Construction graph is built from graphs in Fig. 4.10 as a search space for ACO with the partial edit path $\lambda^{partial} = \Big\{\big(B, \{b, c\}\big), \big(\{A, E, F\}, e\big)\Big\}$. All vertices are connected by edges, but only a few neighboring edges are shown for the readability.

5.1.2 The approach with a complete search space

Our initial approach follows the classical ACO algorithmic scheme for static combinatorial optimization problems: building search space and pheromone trails, constructing a complete edit path, updating pheromone, and stopping conditions. The search space is G_{ants} as presented in the previous section. Ants communicate through pheromone trails where the pheromone amount is laid on the vertices of G_{ants}. Here, we detail the remaining steps with the example of comparing two attributed graphs $G = (V_G, E_G, \mu_G, \xi_G)$ and $H = (V_H, E_H, \mu_H, \xi_H)$ in Fig. 4.10.

Construction of a complete edit path by one ant At each iteration, every ant starts with an empty edit path $\lambda = \{\}$. At each step, it adds a candidate $\nu_i \in G_{ants}$ to the current partial edit path until the edit path is complete which is obtained when λ contains all vertices in G and H.

The ant selects the next candidate regarding the transition probability which is calculated from two factors: the heuristic and the pheromone as in Eq. 5.1.

$$Pr_{\nu_i} = \frac{[\tau_{\nu_i}]^{\alpha} \times [\eta_{\nu_i}]^{\beta}}{\sum_{j=1}^{|cand|} [\tau_{\nu_j}]^{\alpha} \times [\eta_{\nu_j}]^{\beta}}, \tag{5.1}$$

where:

- ν_i is a vertex in G_{ants}, which is a vertex combination in G and H, i.e., $\nu_i = (u, v), u \in V_G, v \in V_H$

- τ_{ν_i} is a pheromone amount of the candidate ν_i, taken from the construction graph G_{ants}
- η_{ν_i} is a heuristic value, or the corresponding cost of the candidate, taken from the cost matrix $\mathbf{C}$
- α, β weight the relative importance between the pheromone and heuristic factors.

Fig. 5.2 gives an example of one ant to build a complete edit path.

Pheromone update When all ants have accomplished constructing the edit paths, the pheromones on each vertex in G_{ants} are updated. It includes two main processes: reinforcement and evaporation. These steps vary depending on each ACO variant. The reinforcement process adds a pheromone amount on the vertices belonging to good edit paths. The evaporation will reduce the pheromone amounts of all vertices in G_{ants}. The reinforcement works as a reward for good edit paths. Meanwhile, the evaporation helps to avoid the convergence to a local optimal solution. Also, it stimulates the natural ant's behaviors: pheromones will be evaporated in the environment according to time.

In subsection 3.2.2, we mention three principal ACO variants, AS, MMAS and ACS. Here, we present the pheromone update for these variants for later usage in the experiments. The update process for each mentioned variant is shown in Table 5.2. For AS, reinforcement is done on each edit path built - meanwhile MMAS and ACS reinforce the pheromone on the best iteration edit path (λ_{itbest}), or the best global edit path (λ_{best}). For ACS, the evaporation process is performed both on local and global scales. For the local scale, it is done for each ant in the colony just after the ant has finished building the complete edit path. For MMAS, after updating, all pheromones will be adjusted in the interval $[\tau_{\min}, \tau_{\max}]$ as in Eq. 5.2.

Variant	Reinforcement	Evaporation	Scale
AS	$\Delta = \begin{cases} \frac{1}{1+c(\lambda_\vartheta)}, & \text{if} \quad \nu_i \in \lambda_\vartheta \\ 0 & \text{otherwise} \end{cases}$	$(1-\rho)\tau_{\nu_i}(t)$	Global
MMAS	$\Delta = \begin{cases} \frac{1}{1+c(\lambda_{itbest})}, & \text{if} \quad \nu_i \in \lambda_{itbest} \\ 0 & \text{otherwise} \end{cases}$		
ACS	$\Delta = \begin{cases} \rho\frac{1}{1+c(\lambda_{itbest})} & \text{if} \quad \nu_i \in \lambda_{itbest} \\ 0 & \text{otherwise} \end{cases}$	$(1-\xi)\tau_{\nu_i}(t) + \xi\tau_0$	Local
		$(1-\rho)\tau_{\nu_i}(t)$	Global

Table 5.2: Updating pheromones on vertices $\tau_{\nu_i}(t+1)$ in G_{ants} for three variants of ACO (with λ_ϑ: edit path built by ant_ϑ; λ_{itbest}: iteration best matching; τ_0: initial pheromone; $\rho, \xi \in [0,1]$: evaporation rates). The local update in ACS is done for all ants at each step.

$$\tau_{\nu_i} = \begin{cases} \tau_{\min} & \text{if} \quad \tau_{\nu_i} < \tau_{\min} \\ \tau_{\max} & \text{if} \quad \tau_{\nu_i} > \tau_{\max} \end{cases} \tag{5.2}$$

Stopping condition The algorithm will stop when it reaches the maximum number of iterations it_{max}. This condition assures that ants can still improve their solution qualities in case of a complex problem or slow convergence.

Local search ACO is known to reach better performance if it is combined with a Local Search (LS) algorithm because both of them are complementary [Dorigo and Stützle, 2019]. ACO searches for solutions at a coarse-grain level. Then, these solutions can be locally fine-tuned by an adequate LS algorithm. The LS is also an example of daemon actions. These are utilized to implement problem-specific or centralized actions that cannot be performed by separate ants [Dorigo et al., 1999]. Therefore, we couple a LS with ACO to enhance its performance for the ExGED problem. At each iteration, when all ant finish building a complete edit path, a LS is applied on the best-iteration matching. We utilize 2-opt and 3-opt for local search.

The 2-opt exchanges two random pairs of vertex mapping in the complete edit path λ. Only valid edit paths from swapping will be considered. For example, in Fig. 5.2, we have $\lambda = \{(A, e), ((E, c), (F, \varepsilon), (B, b), (C, d), (D, a)\}$. We can swap vertex mappings at the 1st and the 2nd positions together. Then, we get new edit path is $\lambda^{swap} = \{(\mathbf{E}, e), ((\mathbf{A}, c), (F, \varepsilon), (B, b), (C, d), (D, a)\}$, and it will obtain a corresponding cost $c(\lambda^{swap})$. If this cost is lower than the initial cost before swapping, it will update λ to λ^{swap}.

For the 3-opt, the principle is similar. However, it has more possibilities for swapping between three pairs of vertex mappings. The λ will be upgraded regarding the swap with the lowest cost. If not, the edit path remains the same. Due to more swap options, the 3-opt is slower than 2-opt. Nonetheless, it can produce a better edit path because more combinations of vertex mappings are created by the 3-opt.

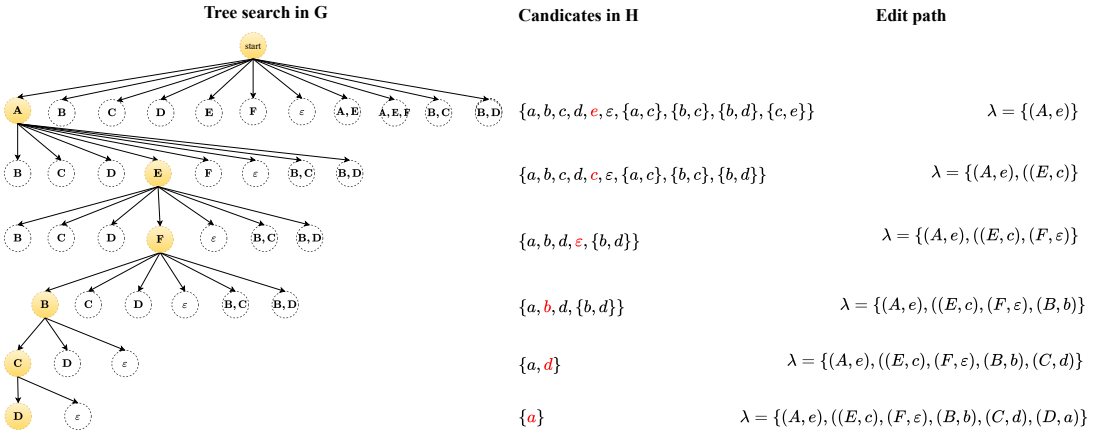


Figure 5.2: Given G and H with $V_G = \{A, B, C, D, E, F\}$ and $V_H = \{a, b, c, d, e\}$, respectively as in Fig. 4.10, the search tree presents all possible remaining candidates when the edit path starts with $A \in G$. The vertices are in yellow and red colors indicating selected vertices in G and H, respectively.

5.1.3 The approach with the search space reduced by neighborhood strategy

Construction of a complete edit path by one ant with neighborhood strategy We have presented the initial approach of ACO for solving ExGED in the previous section. Nevertheless, consideration of all remaining candidates may not be a wise choice for building an edit path due to two following reasons. At one step, one ant selects only one candidate to add to its current edit path. For this choice, all transition probabilities of the remaining candidates needs to be calculated. Therefore, it will be better if the ants can chose the most promising candidates among the remaining ones. To accomplish this purpose, we propose here a strategy based on the neighborhood of the vertex u in G in the previous selected candidate $\nu = (u, v) \in G_{ants}$. From the vertex u, we focus only on candidates that involve the neighbors of u. Then, we will consider these candidates to add to the current edit path. If no neighbor is found, a random choice is made. That is similar if an isolated vertex in G is selected. This neighborhood strategy is less global than the one of the initial approach. But, it is more time-saving, especially for sparse graphs. Notice that in the case of a complete graph, this strategy is useless because all vertices have the same number of neighbors. An example for the improvement with neighborhood strategy is given in Fig. 5.3. Although the number of candidates in H remain the same in Fig. 5.2 and Fig. 5.3, the number of candidates in G in Fig. 5.3 reduces a lot thanks to the neighborhood strategy.

The other steps, such as pheromone update, stopping condition, local search, are done similarly to the initial approach.

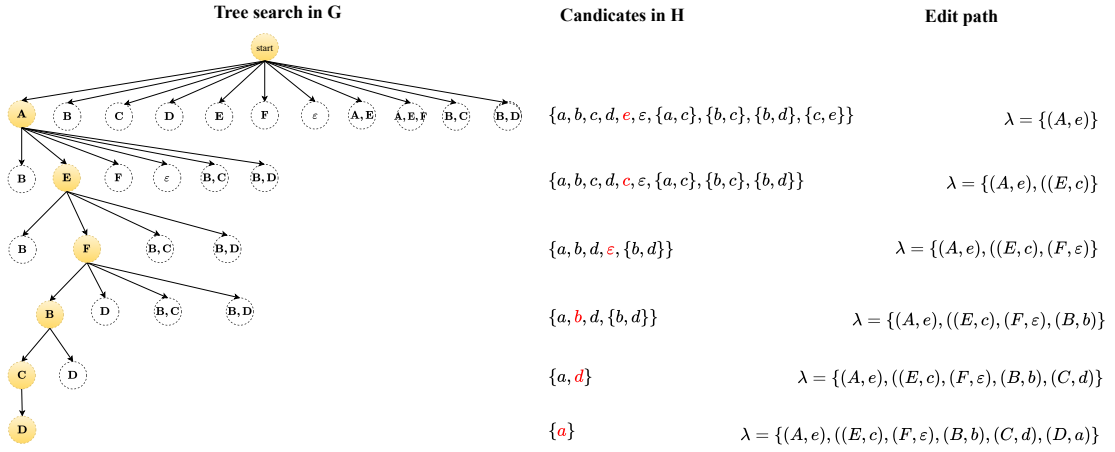


Figure 5.3: Given graph G with the vertex set $V_G = \{A, B, C, D, E, F\}$ and the neighbors of each vertex $Nei_A = \{B, E, F\}, Nei_B = \{A, C, D, E, F\}, Nei_C = \{B\}, Nei_D = \{B, F\}, Nei_E = \{A, B, F\}, Nei_F = \{B, E, D\}$; graph H with the vertex set $V_H = \{a, b, c, d, e\}$ as in Fig. 4.10, a search tree presents all possible candidates with the neighborhood strategy in G when the edit path starts with $A \in G$. Only remaining vertices that are adjacent to the vertex selected previously in G are considered for the next step (random choice is made if the vertex has no neighbor). The vertices are in yellow and red colors indicating selected vertices in G and H, respectively.

5.1.4 General pipeline

After presenting two propositions of ACO for solving ExGED, here we summarize them in a general pipeline as in Fig. 5.4. The inputs are two compared graphs G and H, along with the cost matrix $\mathbf{C}$. The solver is ACO. The output is the edit path with the minimal cost. Besides, we provide a flowchart in Fig. 5.5 to detail the steps of the pipeline in Fig. 5.4.

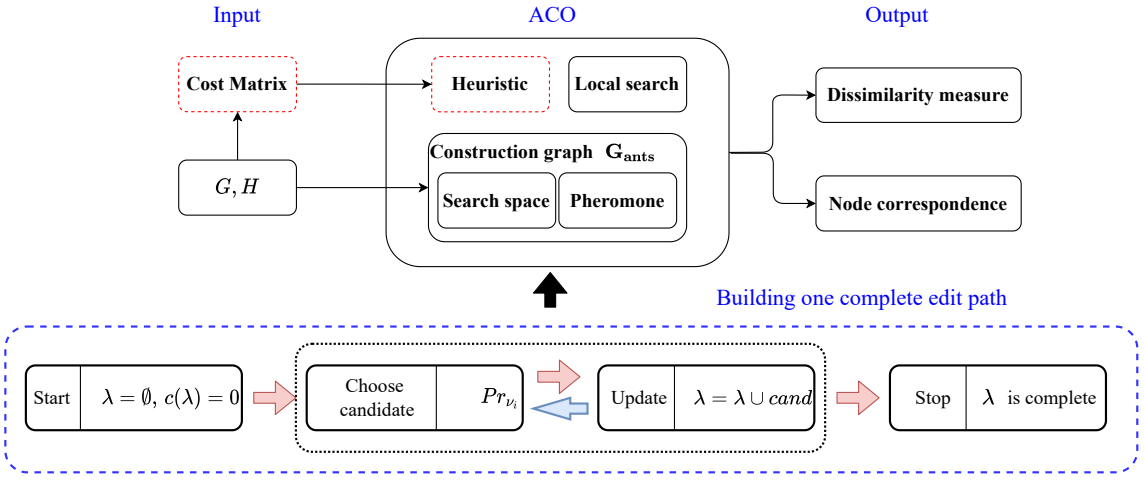


Figure 5.4: A general pipeline of ACO to solve ExGED. The neighborhood strategy can be applied in the process of building one complete edit path. Local search is integrated to enhance the solution quality.

5.2 Experimental setup

In this section, we focus on examining parameters of the proposed approach, including ExGED's and ACO's parameters. For ExGED, the parameter is k, i.e., the maximum merge/split size. For ACO, they are $\alpha, \beta, \rho, pop, it_{\max}$. Besides, typical parameters of each ACO variant are studied. We investigate the impacts of these parameters on the symbol dataset used to construct the graphs for our GM problem. From the numerical results, the values of parameters are set and used for experiments in the subsequent sections.

5.2.1 Data description

5.2.1.1 Dataset

We run the experiments on the symbol dataset, originally from SESYD. This dataset consists of architectural and electrical symbols and can be accessed online [1]. From the original version, various types of noise are added to create deformed symbols by a software called Vectograph. These noises can break an original line into several lines, rotate symbol, or

[1] `http://mathieu.delalandre.free.fr/projects/sesyd/`

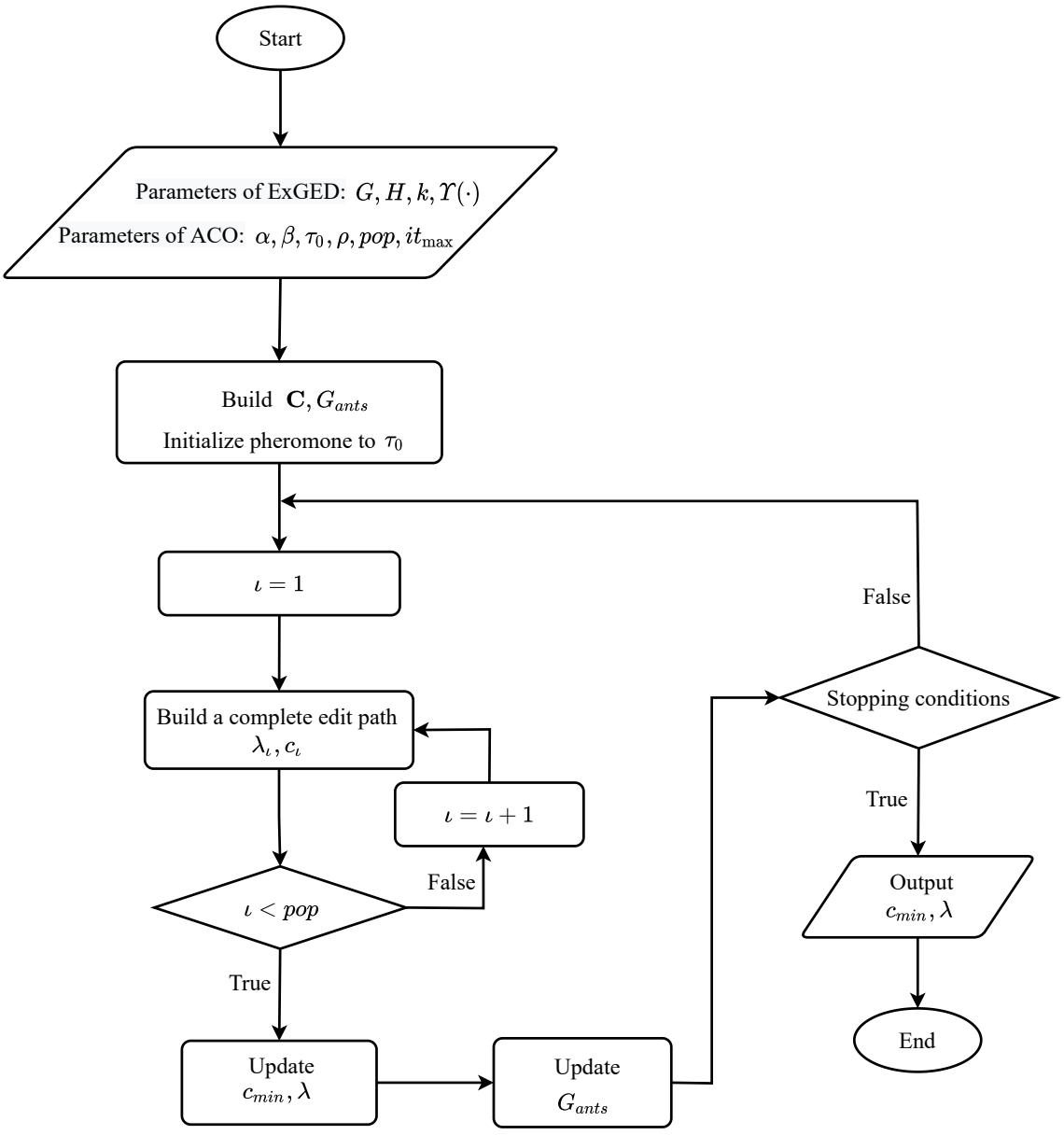
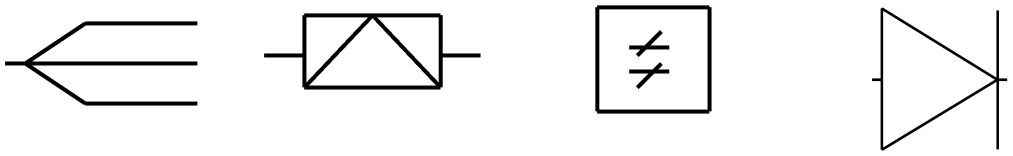


Figure 5.5: A general flowchart of ACO to solve ExGED, where G, H are two input graphs; k is the maximum number of vertices that one vertex can be associated with; $\Upsilon(\cdot)$ is the cost functions; α, β are weights of pheromone and heuristic factors, respectively; ρ is the evaporation rate; pop is the number of ants; $it_{\max}$ is the maximum iteration.

scale the symbol. The new dataset with deformed symbols is freely available via the following link `http://www.rfai.lifat.univ-tours.fr/PublicData/ExGED/home.html`. Fig. 5.6 demonstrates four main types of symbols in the dataset. Table 5.3 presents general information on the dataset used. The graphs are relatively small, but they are fairly dense. Nevertheless, some subsets of the symbol dataset are selected regarding the experiment as shown below.



(i) Lines (ii) Rectangles (iii) Squares (iv) Triangles

Figure 5.6: Four main types of symbols in the symbol dataset.

Value	Model			Noises		
	|V|	|E|	Density	|V|	|E|	Density
max	21.00	34.00	1.00	27.00	43.00	1.00
min	3.00	1.00	0.16	3.00	1.00	0.12
avg.	7.33	9.83	0.47	7.89	10.85	0.44

Table 5.3: General statistical information of graphs from noise-free symbols (models) and noisy symbols in the symbol dataset.

To set up ACO's parameters, we implement the algorithm on four types of symbols to have a general evaluation. Table 5.4 describes more details on these symbols. The dataset is named A henceforth.

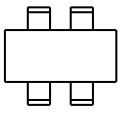
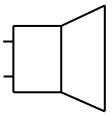
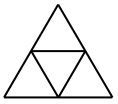
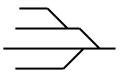
Symbol number / Model	013	032	048	060
				
$\|V_G\|$	21	9	6	7
$\|V_H\|$	25	10	7	9
$\|E_G\|$	34	12	12	6
$\|E_H\|$	41	12	15	10

Table 5.4: Dataset **A** used for the investigation of ACO's parameters, both G and H belonging to the same class.

For the experiments related to the evaluation at the GM level, the dataset in Fig. 5.7 (page 114) is utilized. We use it because it contains symbols with various noise levels that permit us to test the capacity to the proposed method in several situations. When the noise levels are getting higher, the symbols are more noisy, that lead to the deformation of the attributes of vertices/edges as well as of the graph's structure. This dataset is called B to facilitate for later use.

For the evaluation in the classification context, we use another dataset which contains more symbols as shown in Fig. 5.8 (page 115). Moreover, the dataset has several symbols that look similar, which can increase the risk of misclassifying symbols for the algorithms. This dataset is called C to facilitate for later use.

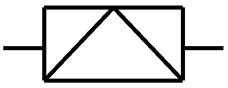
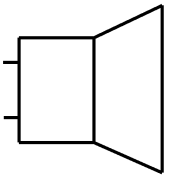
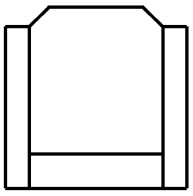
Model	($\|V\|, \|E\|$)						
	H00	**H05**	**H10**	**H15**	**H20**	**H25**	**H30**
002 	(8,13)	(8,13)	(8,13)	(10,13)	(10,13)	(12,16)	(15,15)
064 	(10,17)	(10,16)	(10,17)	(12,22)	(11,163)	(12,19)	(17,29)
032 	(8,11)	(9,12)	(10,12)	(10,12)	(11,12)	(17,20)	(16,17)
012 	(12,18)	(12,18)	(11,16)	(15,22)	(19,23)	(21,25)	(22,21)

Figure 5.7: Information of the dataset **B** used for the evaluation in the GM context provided the Ground Truth (GT) mappings in the Appendix A. H00 represents for noise-free symbols, and others represent for noisy symbols from 5% to 30% compared with the noise-free symbol.

5.2.1.2 Graph representation

To transform symbols into graphs, each line is defined as a vertex, and the relations between lines are edges. Each vertex has its length (l). Each edge has its angle (θ), and type of relation (rel). The length l is normalized regarding the longest line in each symbol, i.e., $l \in [0, 1]$. The rel includes T-Junction (T), Parallel (P), Successive (S), L-Junction (L) and intersection (X). The types of relation are decided based on the angle which is formed between two lines, more details in [Qureshi et al., 2007]. All angles are then normalized in $[0, 1]$ by dividing 180°. The graphs are undirected and contain both numerical and symbolic attributes. We give an example of graph representation in Fig. 5.9 (page 116).

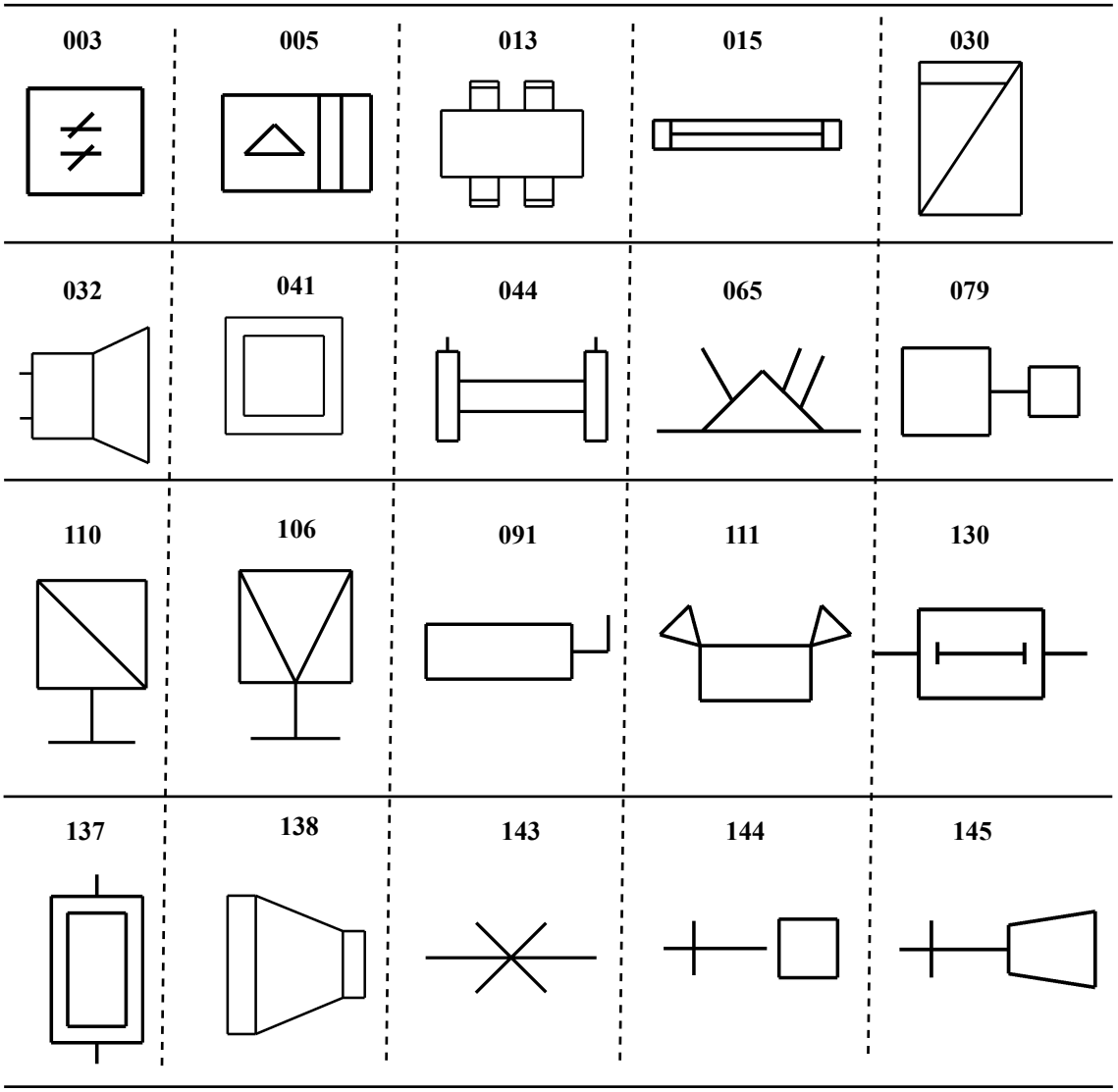


Figure 5.8: Information of the dataset **C** used for the evaluation in the classification context, where $nb_test = 20 * 30 = 600$.

5.2.1.3 Defining cost functions for building cost matrix

Normally, the proposal of cost functions is based on the application knowledge, such as the vertex/edge's attributes and the graph's topology. For vertex 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).

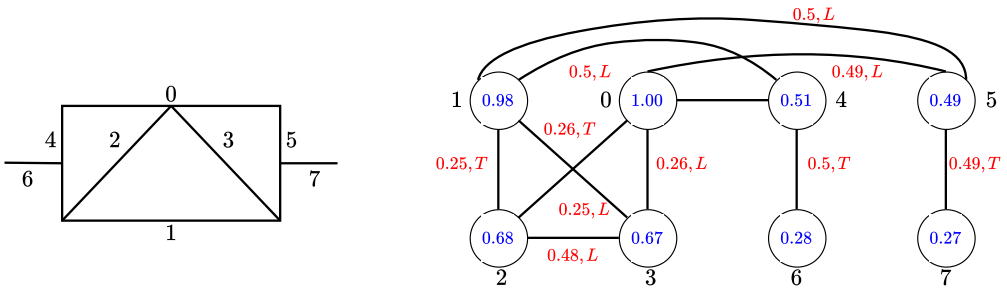


Figure 5.9: The symbol (left) and its graph representation (right), all numerical attributes are normalized to facilitate the calculation. We take an example for the edge relation: the line 5 is connected to the line 7 by the T-junction (T), to the lines 0 and 1 by the L-junction (L).

The values of edge cost functions are also in $[0, 1]$ to be compatible with the vertex cost functions. Details of the cost functions are in Table 5.5 (page 116).

Operations	Vertex operation cost	Edge operation cost
Substitution	$c(u \to v) = \lvert l_u - l_v \rvert$	$c(e_G \to e_H) = \begin{cases} \lvert\theta_{e_G} - \theta_{e_H}\rvert & \text{if } rel_{e_G} = rel_{e_H} \\ 1 & \text{otherwise} \end{cases}$
Insertion	$c(\varepsilon \to v) = l_v$	$c(\varepsilon \to e_H) = 1$
Deletion	$c(u \to \varepsilon) = l_u$	$c(e_G \to \varepsilon) = 1$
Merging	$c(S_{mer} \to v) = \lvert \sum_{u_i \in S_{mer}} l_{u_i} - l_v \rvert$	$c(e \to \varepsilon) = 0, e \in E_{loop}$
Splitting	$c(u \to S_{spl}) = \lvert l_u - \sum_{v_i \in S_{spl}} l_{v_i} \rvert$	$c(\varepsilon \to e) = 0, e \in E_{loop}$

Table 5.5: Cost functions of edit operations for the symbol data set.

5.2.1.4 Define parameter k for splitting and merging

From the theoric analysis in subsection 4.2.2 in Chapter 4, we see a significant increase of the size of the cost matrix $\mathbf{C_{ExGED}}$ regarding k in the general case. Therefore, choosing k is important, which can be based on the priori knowledge to reduce the complexity.

From the observation, we suppose that the S-relation potentially appears when there is noise, but it is not always the case. So, vertices connected by edges with S relations can be candidates for merging or splitting. Initially, we give $k = 3$ for setting up ACO's parameters. Other values of k are also investigated later to see their impacts to the solution quality and the speed.

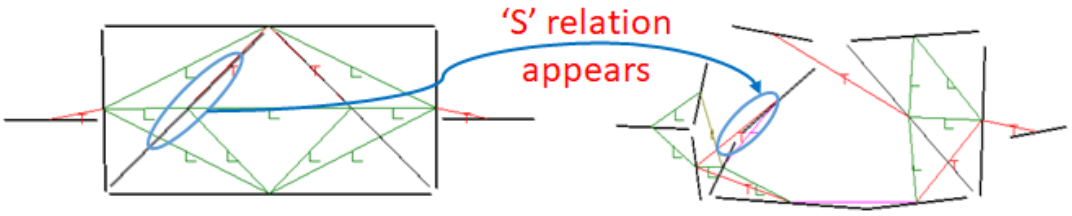


Figure 5.10: The S relation appears when lines in the model (left) are split into several lines in the noisy symbol (right).

5.2.2 Setup for ACO

Because ACO is a meta-heuristic, we need to set up its parameters for each specific problem. Therefore, a section for ACO's parameter tuning is dedicated in order to choose good combinations of them for later experiments. Then, it is essential to examine how the pheromone matrix changes through iterations. That is because it reflects the interactions between ants and the evolution of solution quality. After, we investigate the stability of the results produced by ACO because ACO is not a deterministic algorithm. Finally, we compare efficiency of ACO variants (AS, MMAS, ACS) for tackling ExGED on the symbol dataset.

5.2.2.1 Setting up ACO's parameters

$\boldsymbol{\alpha, \beta}$ are parameters that balance the influence of pheromone and heuristic factors, respectively. We investigate their impacts with various values, i.e., $(\alpha, \beta) \in [(1,0),(0,1),(1,1),(1,2),(1,3),(2,1),(3,1)]$. The 1st tuple implies that the heuristic value is not used for searching the solution. In contrast, the 2nd implies that the pheromone value is not utilized for building the solution. The other tuples employ both pheromone and heuristic factors with different weights. When running experiment for (α, β), we fix other values, such as pheromone evaporation rate (ρ), initial pheromone value (τ_0)), as in Table 5.9. Average results over 30 runs of each variant for the symbol 048 is shown in Fig. 5.6 (page 118), but they are similar for three remaining symbols in Table 5.4 (page 113). In general, utilizing both pheromone and heuristic factors give better solution quality than using only one of them. Here, we observe that good results are obtained generally when $\alpha = 1, \beta = 3$ for AS and MMAS, and $\alpha = 1, \beta = 2$ for ACS. If ACO uses only pheromone factor $(\alpha = 1, \beta = 0)$, the results are usually worse than the other pairs of values. In addition, for more complex symbol such as 013, solutions given by ACS with $\alpha = 1, \beta = 0$ have not been improved during the search. This might be caused by local pheromone update scheme of it. The pheromone did not change significantly during the search phase, and the local search failed to improve the solution. However, this situation happened only with symbol 013, so it is not general. Overall, $\alpha = 1, \beta = 2$ gives competitive results. Therefore, we decide to set $\boldsymbol{\alpha = 1, \beta = 2}$ for ACS, $\boldsymbol{\alpha = 1, \beta = 3}$ for AS and MMAS for later experiments.

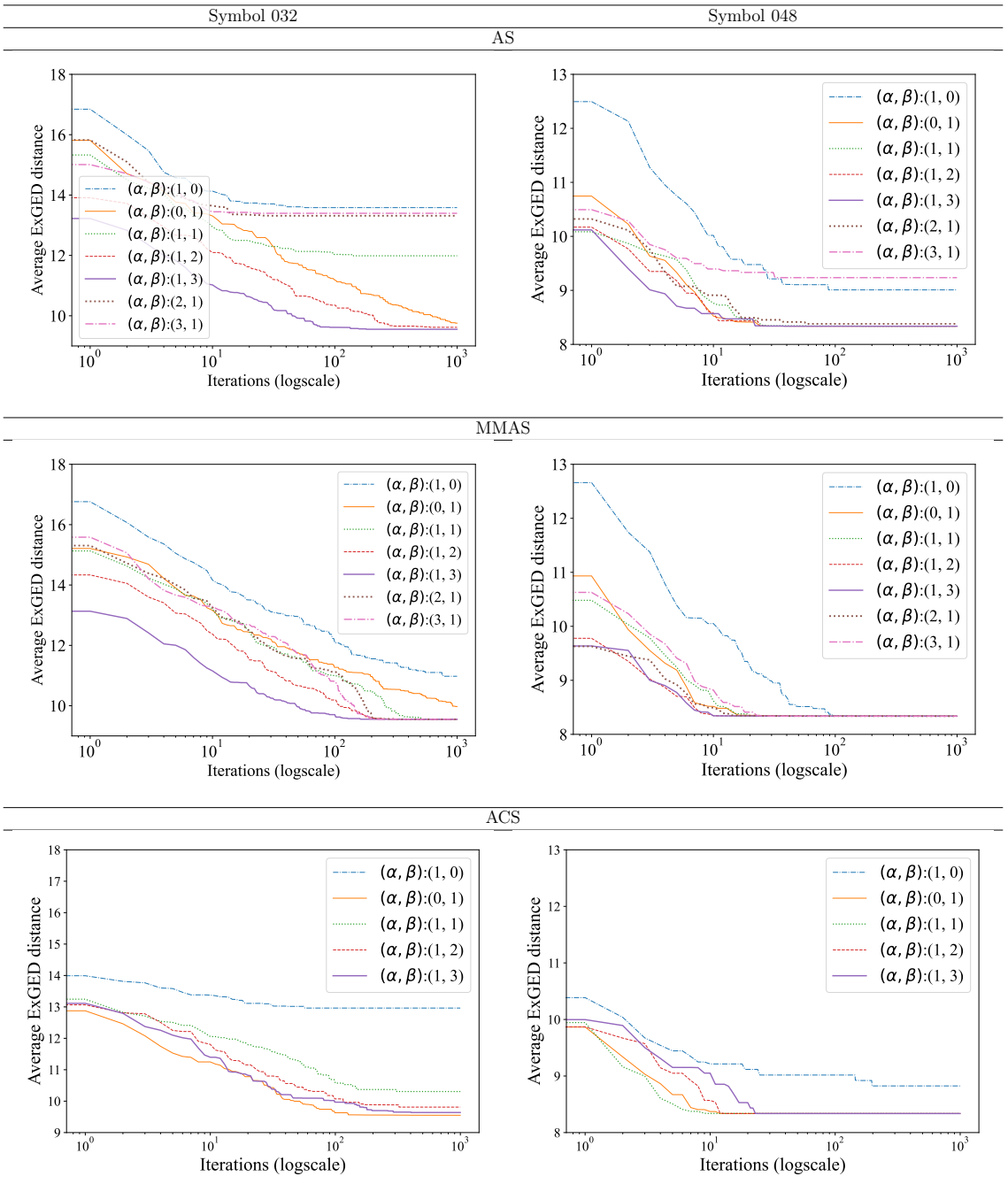


Table 5.6: Influence of α, β to the solution of ExGED problem on the symbol 032 and 048 for three ACO variants. Parameter pairs $(\alpha, \beta) = \{(2, 1), (3, 1)\}$ are not studied for ACS due to its property as shown in Table 3.1.

$\boldsymbol{\rho}$ is also an important parameter of ACO because it links to the learning capacity of ants and also their ability to forget. ρ is always in the interval $[0, 1]$. For this dataset, we examine $\rho = [0.001, 0.01, 0.1, 0.3, 0.9]$. The experiments are done similarly to α, β. The

average result for the symbol 032 are shown in Table 5.7 (page 120) and the results for other symbols also follow this trend. We remark that good ExGED distances are obtained with either $\rho = 0.01$ or $\rho = 0.001$ for almost symbols with different ACO variants. Lower ExGED distances are usually in case of $\rho = 0.3$ or $\rho = 0.9$. That reflects slower pheromone evaporation rate helping to enhance the ExGED distance. From these results, we set $\boldsymbol{\rho = 0.01}$ for AS and MMAS, $\boldsymbol{\rho = 0.001}$ for ACS later on.

***pop*, $\boldsymbol{it_{\max}}$** To study the impact of number of ants (*pop*) to the performance of the proposed approach for ExGED, we test $pop = \{5, 10, 15, 20\}$. The experiment is done on the same data set as before. The results are demonstrated in Fig. 5.11 (page 119). In general, when higher number of ants is used, more expensive the computational time is. In Fig. 5.11, it shows that the computational time grows linearly with the number of ants. AS spends more time to find the solution in comparison with two other variants (MMAS and ACS) in almost all cases. Meanwhile, MMAS and ACS are relatively competitive in terms of computational time. ACS is a bit less time-consuming than MMAS for graphs with bigger sizes, symbol013 for instance. For the maximum number of iterations ($it_{\max}$), we observe that the same ExGED distance is almost achieved from the 200th iteration for simple symbols. For more complex ones, it requires more iterations to improve the ExGED distance. Therefore, we set $pop = 5, it_{\max} = 300$ to balance the simple and the complex symbols.

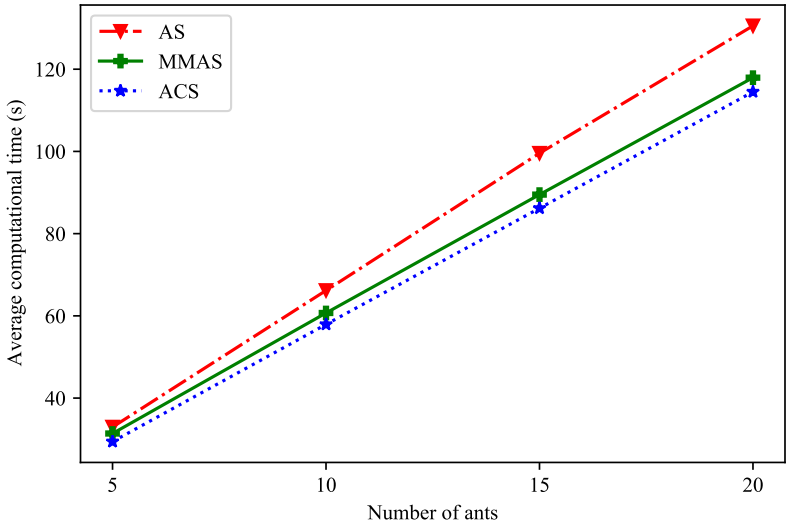


Figure 5.11: Average computational time of three ACO variants regarding different number of ants (*pop*) for solving ExGED on symbol dataset.

$\boldsymbol{q_0}$, $\boldsymbol{\xi}$ for ACS Similarly, we examine the influence of exploitation/exploration threshold (q_0), and local evaporation rate (ξ) of ACS on the solution quality. The results show that these parameters have little impact on the solution quality of the surveyed data as shown

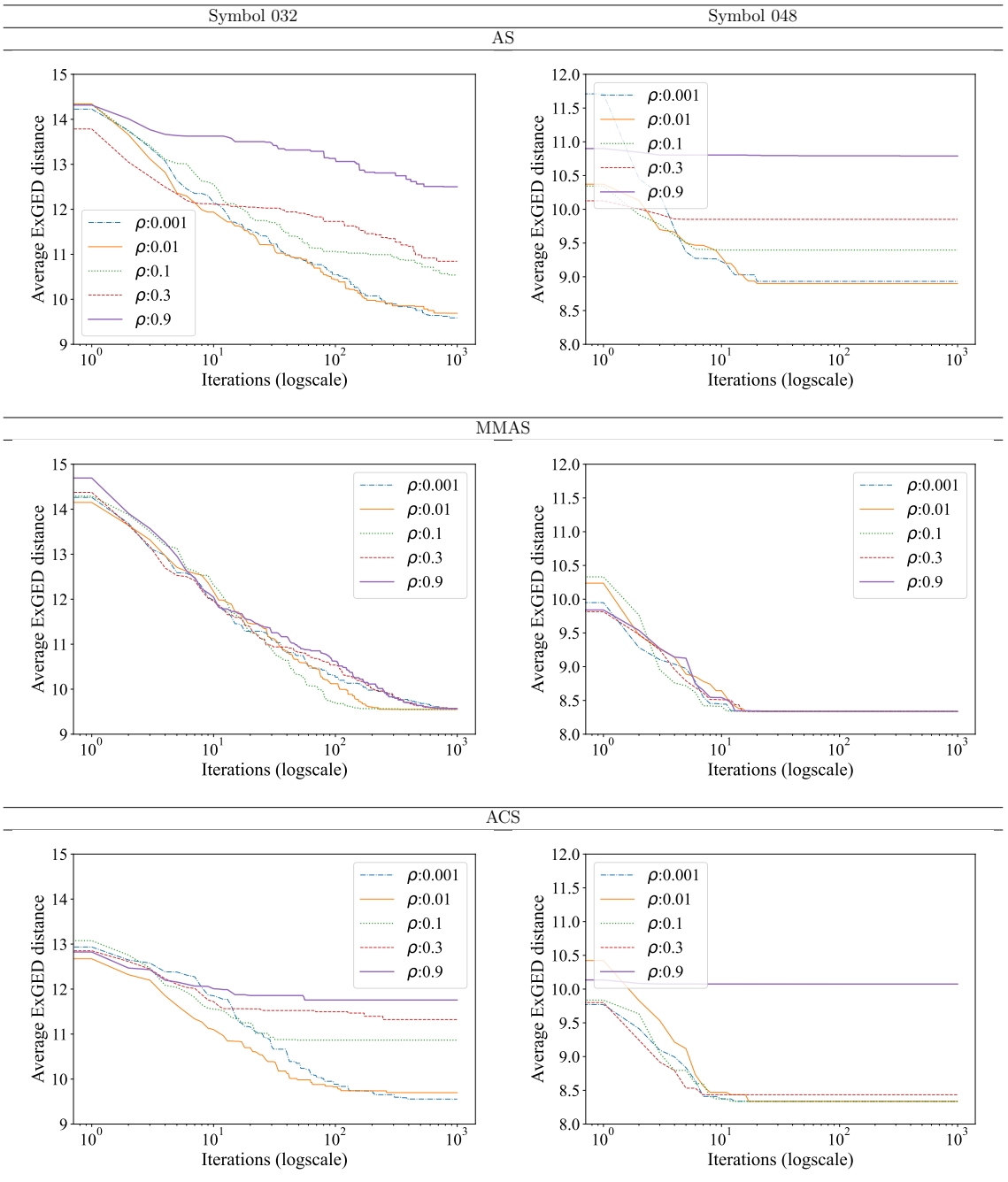


Table 5.7: Influence of ρ to the solution of ExGED problem on the symbol 032 and 048 for three ACO variants.

in Table 5.8 (page 121). Nevertheless, better results are acquired with $q_0 = 0.7, \xi = 0.01$. Therefore, we consider these values for q_0 and ξ later on.

To summarize, all parameters are set as in Table. 5.9 (page 121) for subsequent experiments on the symbol dataset.

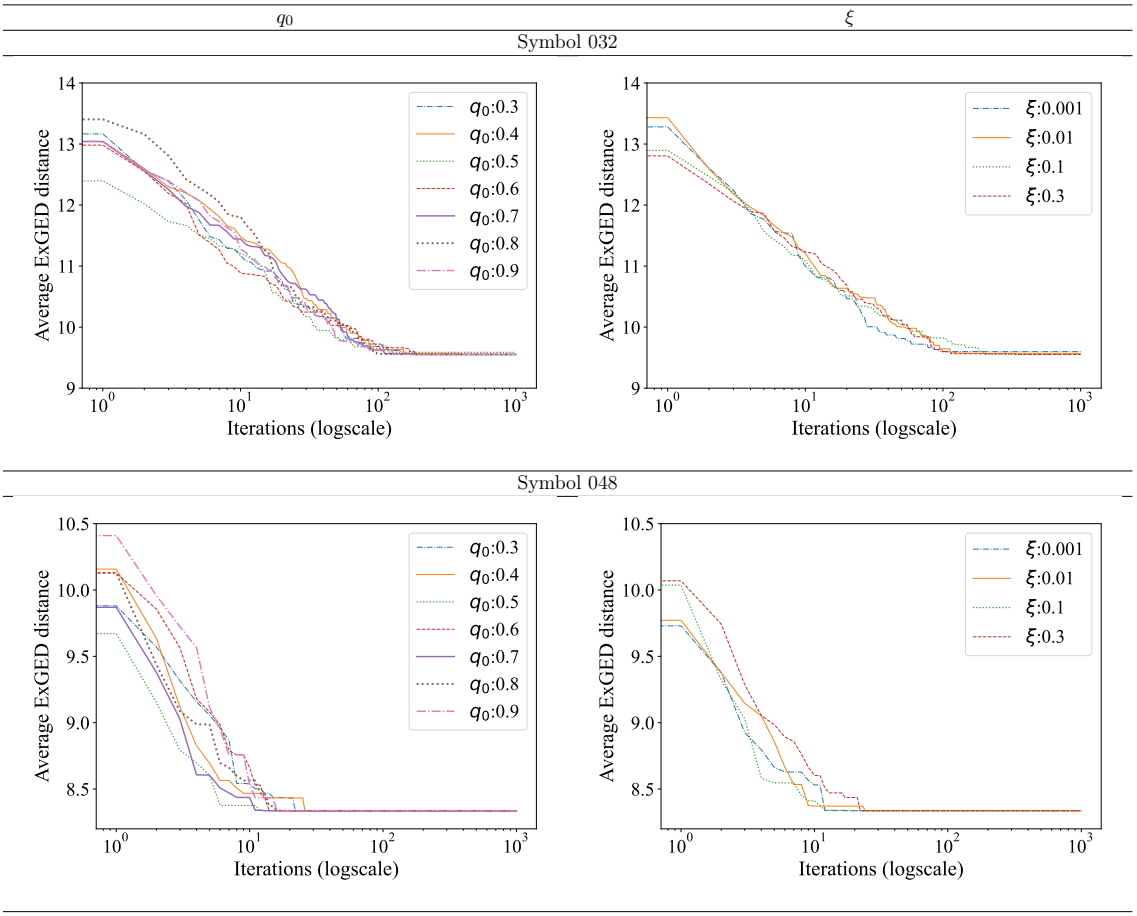


Table 5.8: Influence of q_0, ξ of ACS to the solution of ExGED problem on the symbol dataset.

Variant	(α, β)	ρ	τ_0	pop	$it_{\max}$	$[\tau_{\min}, \tau_{\max}]$	q_0	ξ
AS	(1, 3)	0.01	0.9	5	300	–	–	–
MMAS	(1, 3)	0.01	1.0	5	300	[0.01, 1.0]	–	–
ACS	(1, 2)	0.001	0.9	5	300	–	0.7	0.01

Table 5.9: Summary of setting up values for parameters related to ACO variants for experiments on the symbol dataset afterwards. Values for pheromone bounds $\tau_{\min}, \tau_{\max}$ are studied, but they don't not influence a lot to the solution, and they are set between 0.1 and 1.0.

In the section, the experiments have been done to find good combinations for later experiments. Nevertheless, these experiments have been implemented on a small dataset and the orders in which parameters have been considered can have an influence on the choice of values. But other tests on the full database will maybe confirm these preliminary choices.

5.2.2.2 Influence of local search algorithms to ACO's solution quality

As mentioned in section 5.1, LS can help to enhance the solution quality produced from ACO theoretically. Hence, we study how the solution quality will be affected with and without LS. The results are demonstrated on the symbol dataset as in Table 5.10 (page 123). It shows that the algorithm is hard to obtain a qualitative solution without LS. With LS, a lower ExGED distance is achieved within fewer iterations. It is true in the case of AS, where LS aids a lot to achieve smaller ExGED distance. Overall, 3-opt LS gives more robust results than 2-opt LS and without LS. Therefore, we use 3-opt LS for later experiments.

5.2.2.3 Studying the interaction between ants via the evolution of the pheromone matrix

In nature, ants interact together to find the best path from their nest to the food source. It is done by laying their chemical substance (pheromone) on the paths. These interactions help guide them to a shorter path [Dorigo and Gambardella, 1997]. For artificial ants in ACO, this property is simulated by pheromone values (τ). They mimic the real interaction through the process of pheromone update (reinforcement and evaporation). Therefore, it is essential to study this aspect of ACO. It shows how the ant colony cooperates to get good edit paths for the ExGED problem. Also, it demonstrates the evolution of the solution over iterations.

We show the communication between ants via the pheromone matrix which is the matrix representation of the construction graph G_{ants}. The pheromone matrix has the same size as the cost matrix $\mathbf{C}$, and we set a value of 0 for forbidden mappings. In the beginning, each position in the pheromone matrix has the same value, equivalent to the initial pheromone τ_0, except for the forbidden mappings. After each iteration, these values are changed according to solutions/edit paths constructed by ants. A position with a high pheromone value indicates that it has been a part of the preceding good matchings. In the end, positions with the highest values show good edit paths that ants have passed.

To study this aspect, we take the same dataset for setting up ACO's parameters. We run 30 times for each symbol/each variant. The average results are displayed. Based on the numerical results, we observe two cases: the good paths are visible on the pheromone matrix, the good paths are non-observable. These cases are shown in Table 5.11 (page 124) for symbol 032 (observable) and in Table 5.12 (page 125) for symbol 013 (non-observable), respectively. For symbol 032, the pheromone matrices of three variants at the 300th iteration are similar to some extent. Specifically, the pheromone values are high at positions that belongs to the good edit path, for example $(n1, n10), (n2, n11), (n8, n19), (n9, eps)$. In contrast, the pheromone matrices of three variants at the 300th iteration are very different for symbol 013. The traces of good edit paths are not explicit in this case.

In addition, the pheromone matrix produced by ACS is less similar than the ones yielded by AS and MMAS because ACS has a very different working principle compared to AS and MMAS. ACS has more visible edit paths on the pheromone matrices maybe because of its pheromone update scheme. The pheromone values of ACS decrease slowly due to its local pheromone update. Inversely, pheromone matrices of AS have much higher

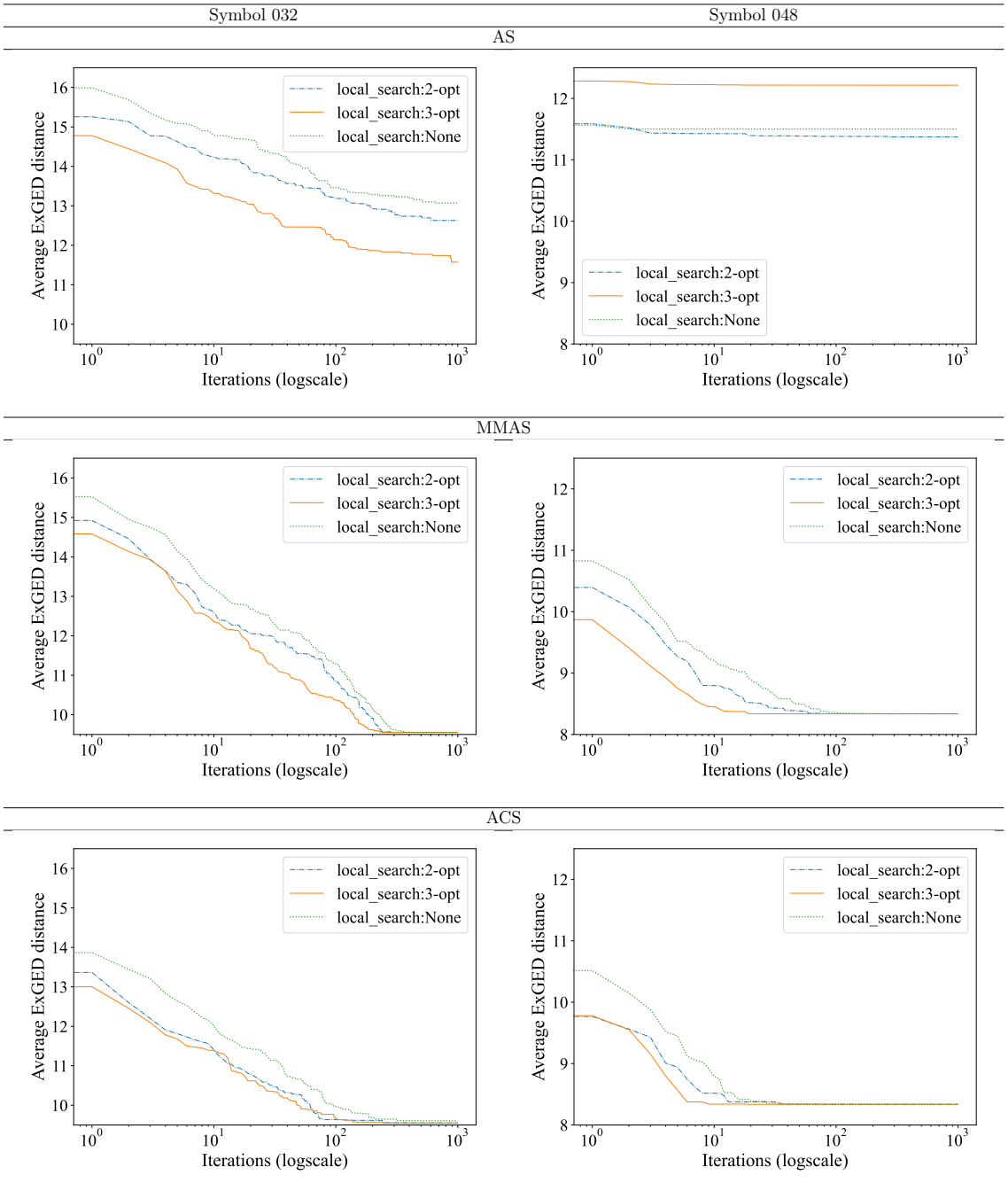


Table 5.10: Influence of local search to the solution of ExGED problem on the symbol 032 and 048 for three ACO variants.

values. Because there is no bound on the pheromone values, all ants reinforce pheromone on their edit path. For MMAS, all pheromone values are always in predefined bounds.

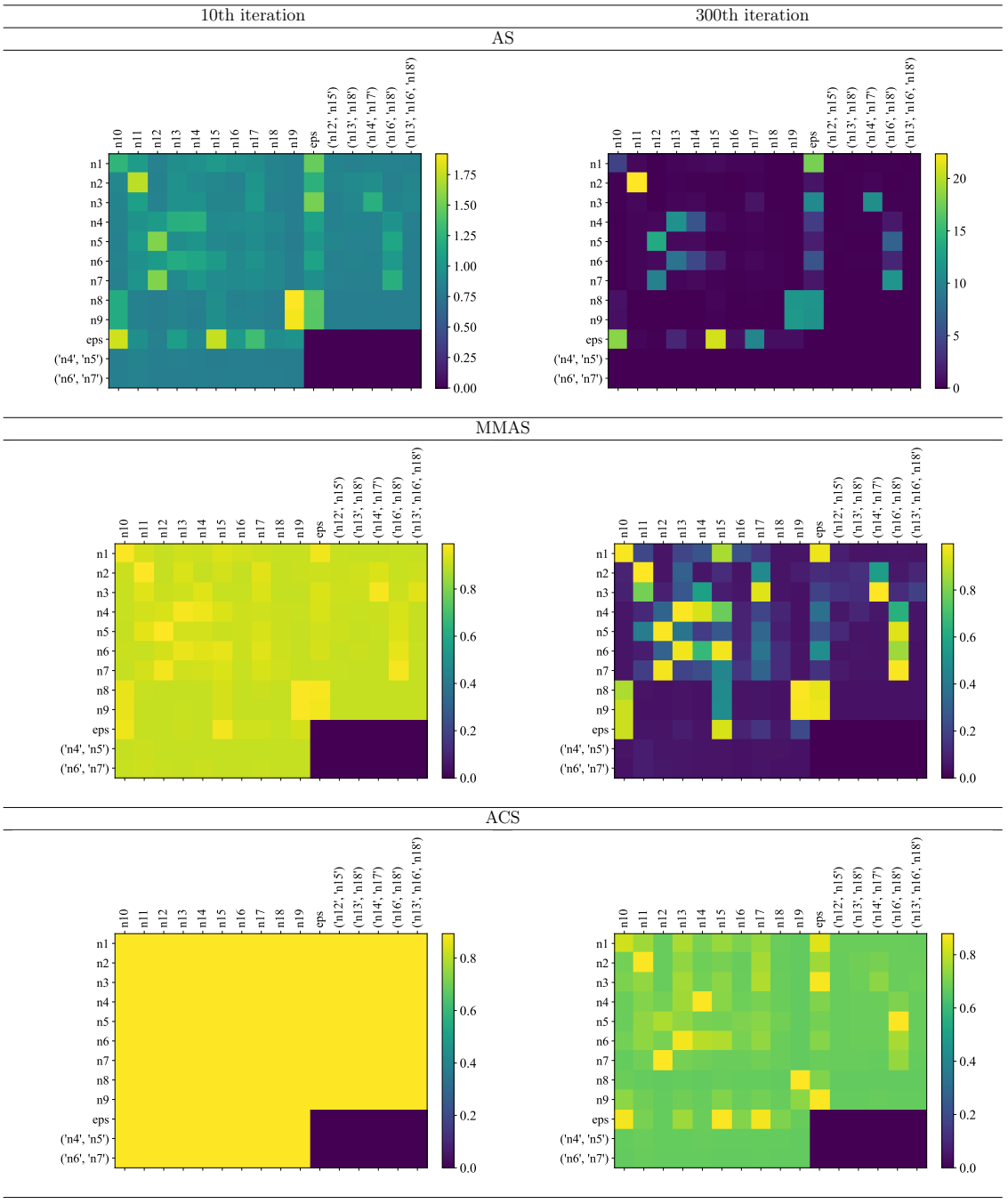


Table 5.11: Interaction between ants demonstrated via the evolution of the pheromone matrix over iterations for three ACO variants on the symbol 032. Rows' and columns' labels each matrix are vertex candidates of two compared graphs, where 'eps' represents virtual vertices. The positions of value 0 are equivalent to forbidden mappings in ExGED. In the beginning (the 1st iteration), all positions have the same values.

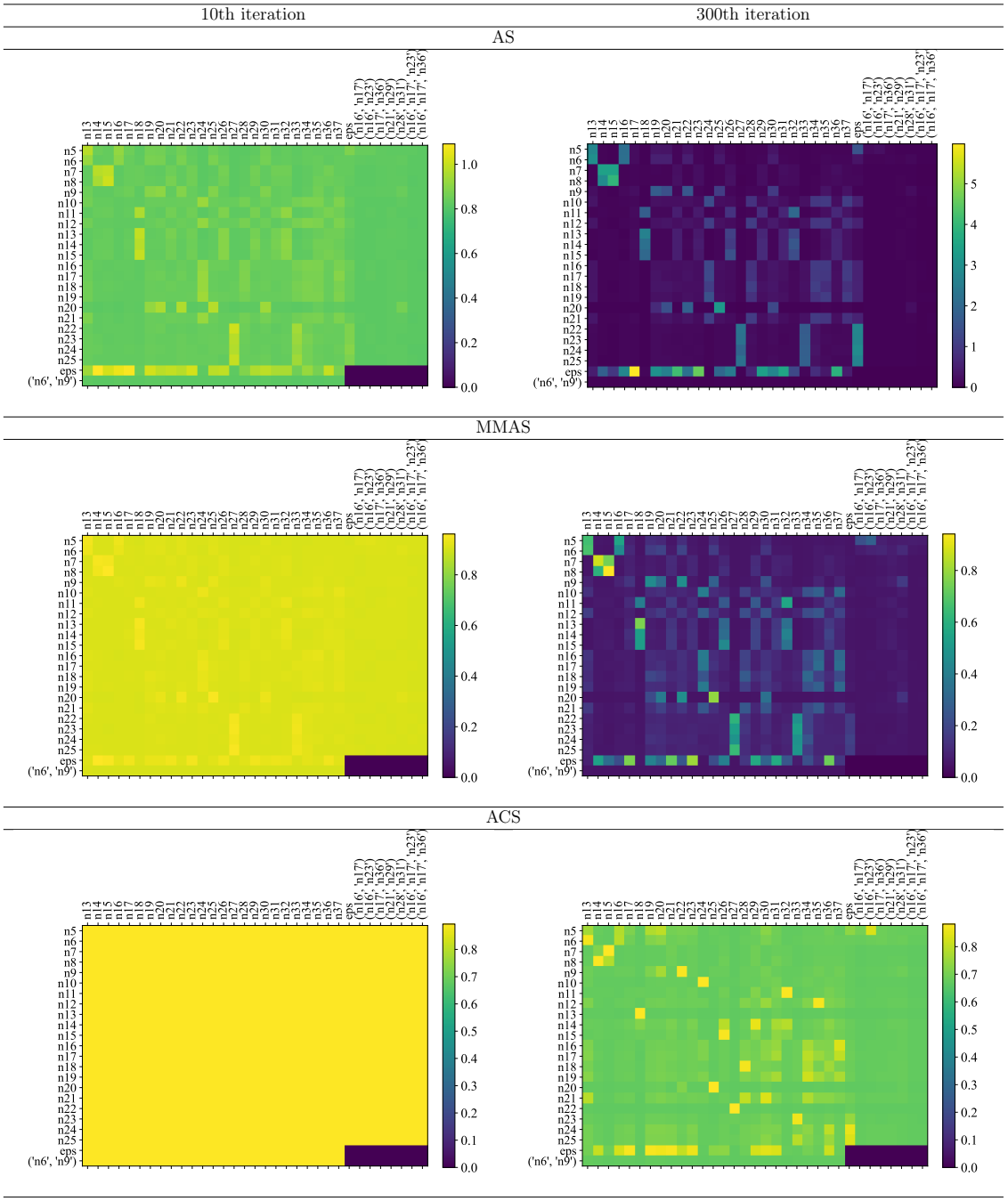


Table 5.12: Interaction between ants demonstrated via the evolution of the pheromone matrix over iterations for three ACO variants on the symbol 013. Rows' and columns' labels of each matrix are vertex candidates of two compared graphs, where 'eps' represents virtual vertices. The positions of value 0 are equivalent to forbidden mappings in ExGED. In the beginning (the 1st iteration), all positions have the same values.

5.2.2.4 Demonstrating the non-deterministic property of ACO

ACO is a non-deterministic algorithm. As a consequence, the solution produced by it may not be stable. ACO can provide different solutions during multiple runs [Dorigo and Stützle, 2019]. Therefore, it is necessary to check this property of ACO on each dataset. Due to a lack of datasets for multivalent GM, we study the stability only on the symbol dataset. Note that this is just a simulation to show the instability of solutions yielded by ACO. We design the experiment with the dataset A in Table 5.4. The parameters for each ACO variant are set as in Table 5.9. We run 30 times for each comparison performed by each variant. The results are shown in Fig. 5.12.

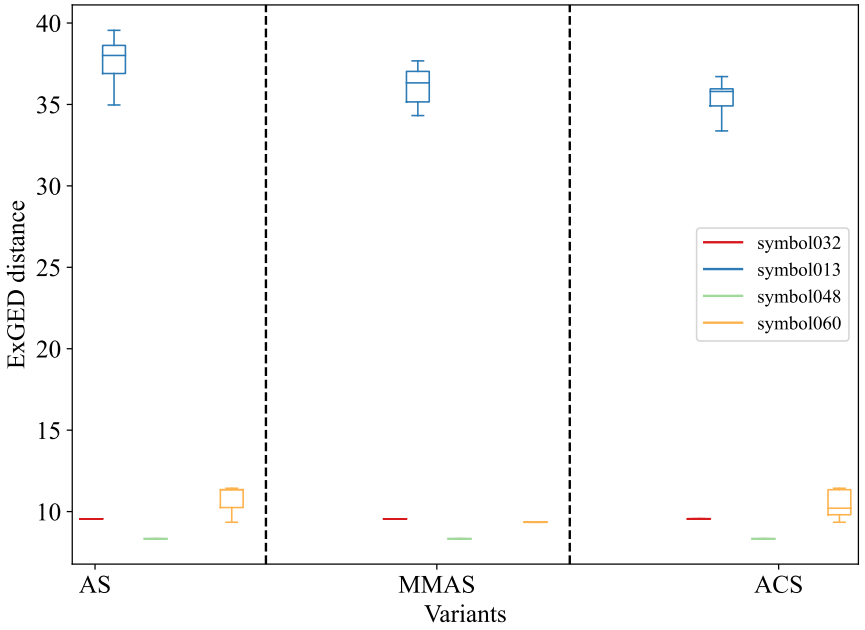


Figure 5.12: Box plot to verify the performance stability of three ACO variants, each experiment for each model, each variant is run 30 times.

The box plot shows that three variants obtain the same results for simple symbols, such as 032 and 048. They produce the same ExGED distance in these cases. In addition, the costs at the 300th iteration of all variants during 30 runs are all equal for symbols 032 and 048. Maybe ACO variants find the optimal solution in these cases, or they are trapped into the same local solution. The ExGED distance gap is higher between the three variants for more complex symbols. For symbol 060, MMAS is the most stable, followed by ACS and AS, respectively. For symbol 013, MMAS and ACS achieve almost similar costs. But, MMAS has less fluctuation than ACS. AS gets the worst performance for the symbol 013. Generally, MMAS is more stable than AS and ACS on the experimented small dataset.

5.2.2.5 Comparison between popular variants of ACO

In this section, we compare the performance of AS, MMAS, and ACS on the ExGED problem for the symbol dataset. We take four models from the dataset. For each model G_{model}, we add noise to create deformed symbols G_{noise}. The levels of noise are in $\{5\%, 10\%, 15\%, 20\%, 25\%, 30\%\}$ in comparison with the original model. This is the dataset B presented previously. Thus, we do graph matching between the model G_{model} with each noise G_{noise} (Table 5.13 (page 128)). We have $4 \times 6 = 24$ comparisons.

We evaluate the results in terms of ExGED distance, computational time, and low-level matching. For two initial comparisons, we use both numerical data and statistical tests. For the 3rd comparison, we verify the vertex mappings derived from each variant for each pair of graphs. Then, we compare with the Ground Truth vertex mappings (GT). These GT mappings are defined manually and presented in detail in Appendix A. The costs of the GT mappings are deduced in the following way. A cost matrix is built similarly as $\mathbf{C}$, but with the true candidates for splitting and merging. The cost functions are the same as in Table 5.5. The estimated edge costs are also integrated with the vertex mapping costs. Then, the cost of the GT mapping is the sum of each position of true vertex mapping in the matrix. The last assessment is to verify whether or not a lower ExGED distance implies a meaningful matching. In other words, we evaluate both quantitative and qualitative aspects.

The ExGED distance and computational time of each ACO variant are presented in Fig. 5.14 (page 129). It shows that the ExGED distances derived by MMAS are closest to GT distances, followed by ACS, and AS, respectively. In detail, three variants achieve the same distances when dealing with symbols of low noise levels such as 5, 10. When the noise levels grow up, there are more gaps between them. Moreover, the ExGED distances obtained by MMAS and ACS are competitive in several cases, especially with high noise levels. Regarding computational time, AS is the most time-consuming variant. MMAS spends more time than ACS to compute the solution. So, ACS can be a good candidate among the three variants due to a good balance of performance and computational time.

We verify that by performing a statistical test. Before that, we verify the normal distribution of costs and computational times of the three variants. The results show that they do not follow a normal distribution (Fig. 5.15 (page 129)). Therefore, the Kruskal-Wallis test is used instead of ANOVA. The null hypothesis is: there is no difference between the cost/computational time among ACO variants with a significant level of 95%. The statistical results give $p \geq 0.05$ which confirms that the difference between these variants is not statistically significant in terms of ExGED distance and computational time. So, the quantitative evaluation gives no distinct result between the three variants. However, MMAS is the one that gives the closest ExGED distance to the GT. To be more certain and general, we examine the low-level mappings yielded by these variants in the next paragraph.

When looking at Fig. 5.14, there are model_noise for which ExGED distances produced by ACO variants are higher than the GT distance. For example, it is the case when the noise levels are 20 for symbol 032 or 25 for symbol 012. We verify the low-level matching (vertex-to-vertex mappings) between symbol_noise obtained from variants, and we compare with the GT mappings. We divide into two cases to evaluate. The first is whether a similar

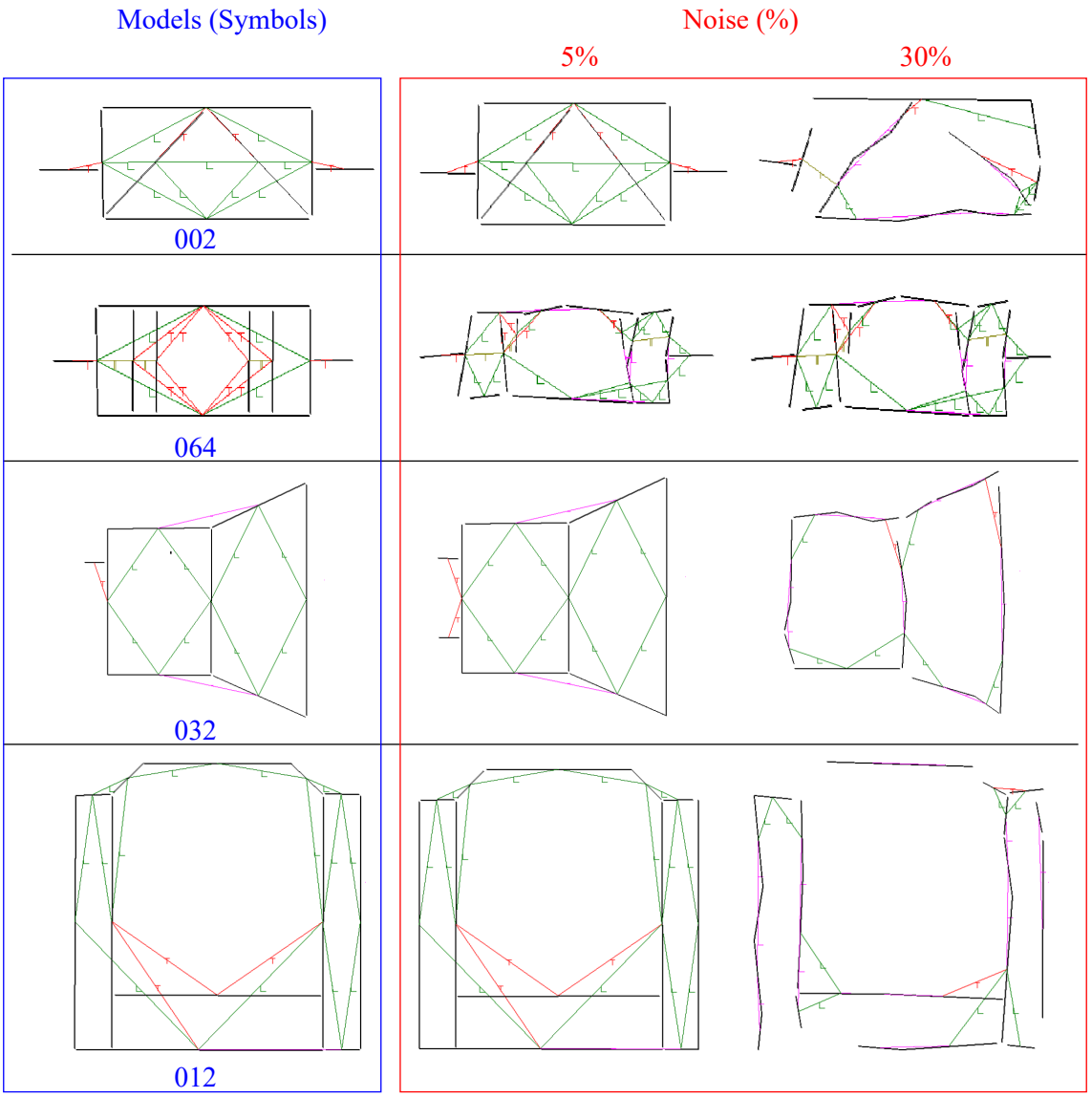


Figure 5.13: Some examples of models and its noisy symbols (at 5% and 30% compared with the original symbols) to serve for comparing ACO variants' performance, higher noise level results in more noisy symbols.

distance of three variants indicates the unique mapping and the same with the GT. The second is when ACO variants acquire different distances, whether or not we select the lowest one. Then, we compare the mapping corresponding to that distance with the GT mapping. We have the following remarks:

- the same distances do not mean similar mappings
- distance produced by ACS and MMAS, such as 12_25, 32_20, can be lower than one of GT, but not implying an expected mapping in the GT

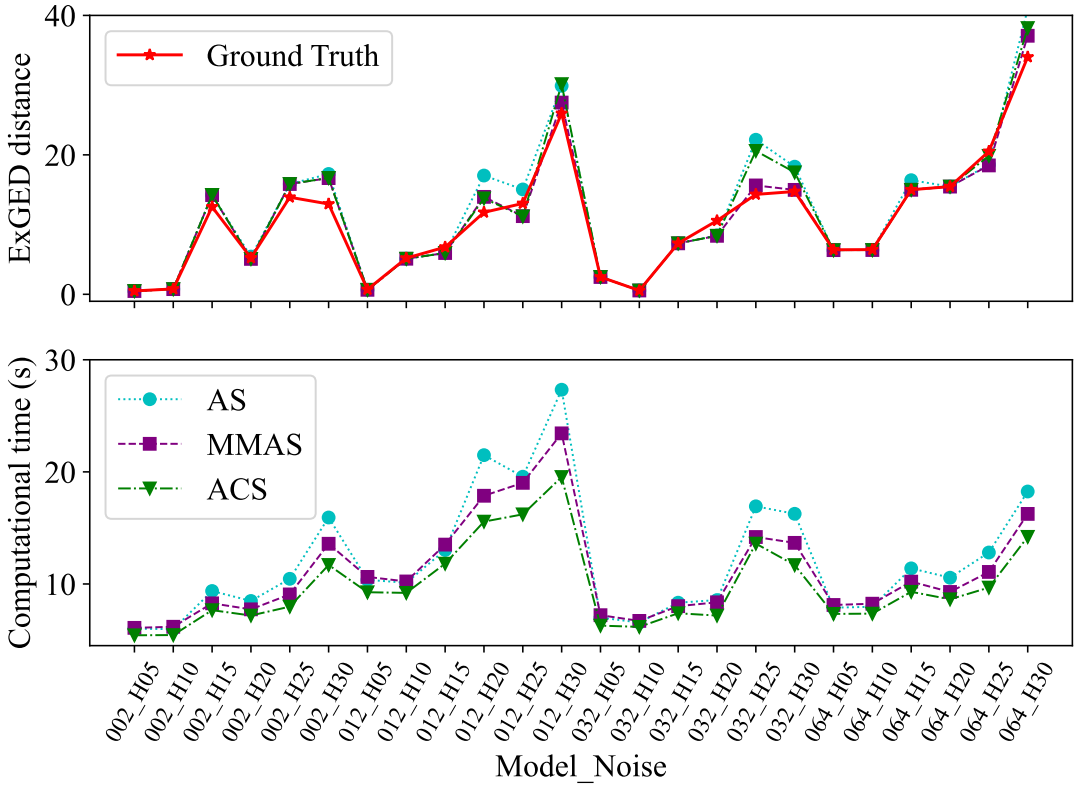
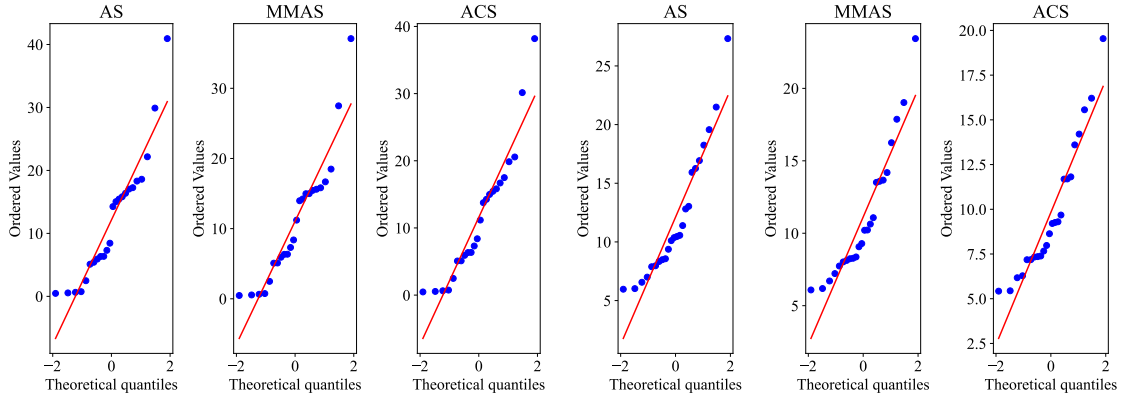


Figure 5.14: Results of three ACO variants for ExGED on the symbol dataset in terms of ExGED distance and computational time.



(i) ExGED distance (ii) Computational time

Figure 5.15: Verify normal distribution of ExGED distance and computational time derived by three ACO variants for ExGED on the symbol dataset before performing the statistical test.

- more similar parts of mapping are found with pair of model and its noisy symbols at low levels such as 5, 10, and 15
- pair between model 032 and its noisy symbols 032 (032_10) is the unique case where mappings produced by ACS and the GT mapping are the same
- MMAS gives the closest mapping to GT mapping among three variants

The symbols used in these experiments are symmetric. Therefore, it increases the difficulty of the problem because it is hard to distinguish the local structure of vertices. Also, the cost matrix **C** does not reflect the global topology of the graphs because the edge relations are not computed precisely to reduce the problem complexity. Consequently, a similar ExGED distance is possible for different mappings. Besides, the symbols are so distorted that the GT solution becomes infeasible to find. In these cases, the lowest ExGED distance might not indicate an expected GT mapping. In another aspect, it is logical that a better similarity of mapping between ACO variants and GT is achieved when models are less deformed (low levels of noise). Among three variants, MMAS gives the smallest ExGED distances more frequently, which confirms the resemblance of its mappings with the GT mappings.

5.2.3 Influence of the neighborhood strategy to reduce the search space

In section 5.1, we proposed two versions of ACO for ExGED: one with a complete search space and another with a reduced search space by neighborhood strategy. Theoretically, the second version aims at accelerating the computational time for ACO. However, this strategy can be useless if the compared graphs are complete. We study these hypotheses on the symbol dataset for two versions. Due to the efficiency of MMAS demonstrated previously, we employ it here for the experiments. We set 'completeMMAS', 'reducedMMAS' be the MMAS without and with the neighborhood strategy, respectively. The dataset B is used for the evaluation. It contains more complex symbols with various noise levels. We analyze the numerical results in terms of computational time (s) and solution quality.

Fig. 5.16 (page 131) presents results with regard to two evaluation criteria. The beneath sub-figure shows that the reducedMMAS is less time-consuming than the completeMMAS. It spends time almost equally to pair the models with their noisy symbols (violet dashed line). In contrast, the completeMMAS spends more time on more complex symbols, such as 012, and it is more fluctuated. That partly confirms our hypothesis about the second version related to the computational time. The sub-figure at the top says that the reduced-MMAS produces as competitive ExGED costs as the completeMMAS. Even in some cases, it yields closer costs to the GT than the completeMMAS, notably in pairs of model 012. To be more confident, we also inspect the low-level matching provided by the two approaches and compare them with the GT mapping.

From the numerical results, we observe that, the lower noise levels are, more similarity between the mapping of GT and MMAS will get. When the noise level from 20, the vertex mappings from completeMMAS and reducedMMAS are more diverse, and less similar to these of GT. The symbols at these levels are highly distorted, that results in strong

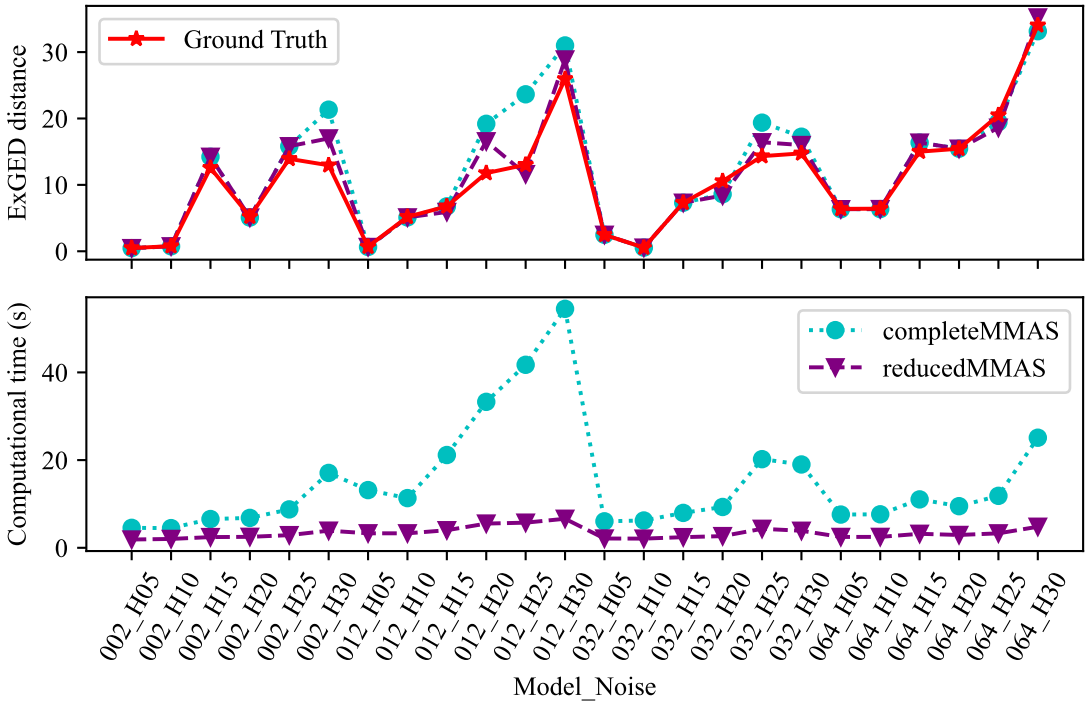


Figure 5.16: Comparison MMAS with and without the neighborhood strategy in terms of ExGED distance and computational time (s).

deformations from the original model. This leads to many changes in the structure of the symbols. Consequently, there are more differences between the graphs from the models and the noisy symbols. In some situations, one line is split into four smaller lines. By setting $k = 3$, we cannot expect to obtain the same mapping as GT.

Another difficulty is the symmetric topology of models used in the experiment. This problem has been mentioned in the comparison of three ACO variants. These reasons can explain the difference between the GT mapping and the MMAS mapping. Between completeMMAS and reducedMMAS, their vertex mappings are relatively similar for noise levels under 20. Nevertheless, the mappings from completeMMAS are closer to the ones of GT to some extent. The neighborhood strategy restricts the search area, so it can miss some important non-neighborhood candidates when building the solution. We give one example for model 032 as in Fig. 5.17 and in Table 5.13. The vertex mappings are quite similar between them. Especially, the mapping of completeMMAS is identical to that of GT in the case of 032_H10 and 032_H15. For reducedMMAS, it achieves the same ExGED distances as completeMMAS and GT. But its vertex mappings are slightly different. The symmetric topology of the models and the estimated edge costs can explain this situation. More importantly, we note that both completeMMAS and reducedMMAS succeed in matching the split lines with their original line in three cases.

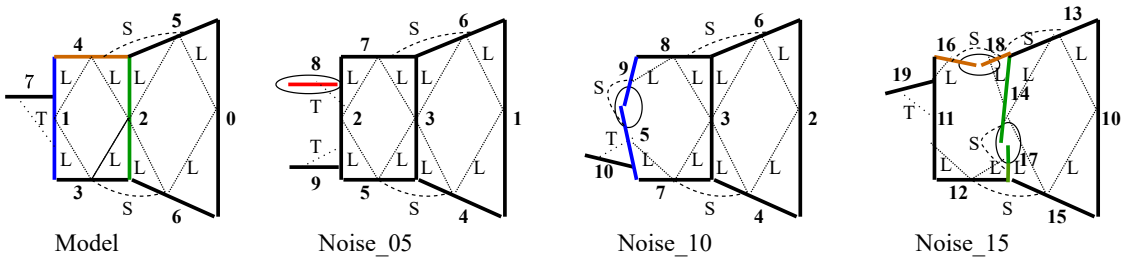


Figure 5.17: Drawing of vertices (in bold) and edges (dashed lines) on the symbol 032 and its noise levels. Ellipses indicate noise positions, same noise position illustrated with the same color.

	032_H05	032_H10	032_H15
GT	('n0', 'n1'), ('n1', 'n2'), ('n2', 'n3') , ('n3', 'n5'), ('n4', 'n7'), ('n5', 'n6'), ('n6', 'n4'), ('n7', 'n9'), ('eps', 'n8')	('n0', 'n2'), ('n1', ('n5', 'n9')), ('n2', 'n3'), ('n3', 'n7'), ('n4', 'n8'), ('n5', 'n6'), ('n6', 'n4'), ('n7', 'n10')	('n0', 'n10'), ('n1', 'n11'), ('n2', ('n14', 'n17')), ('n3', 'n12'),('n4', ('n16', 'n18')), ('n5', 'n13'), ('n6', 'n15'), ('n7', 'n19')
completeMMAS	('n0', 'n1'), ('n1', 'n2'), ('n2', 'n3'), ('n3', 'n7'), ('n4', 'n5'), ('n5', 'n6'), ('n6', 'n4'), ('n7', 'n9'), ('eps', 'n8')	('n0', 'n2'), ('n1', ('n5', 'n9')), ('n2', 'n3'), ('n3', 'n7'), ('n4', 'n8'), ('n5', 'n6'), ('n6', 'n4'), ('n7', 'n10')	('n0', 'n10'), ('n1', 'n11'), ('n2', ('n14', 'n17')), ('n3', 'n12'), ('n4', ('n16', 'n18')), ('n5', 'n13'), ('n6', 'n15'), ('n7', 'n19')
reducedMMAS	('n0', 'n1'), ('n1', 'n3'), ('n2', 'n2'), ('n3', 'n5'), ('n4', 'n7'), ('n5', 'n6'), ('n6', 'n4'), ('n7', 'n9'), ('eps', 'n8')	('n0', 'n2'), ('n1', ('n5', 'n9')), ('n2', 'n3'), ('n3', 'n6'), ('n4', 'n8'),('n5', 'n7'), ('n6', 'n4'), ('n7', 'n10')	('n0', 'n10'), ('n1', 'n11'), ('n2', ('n14', 'n17')), ('n3', ('n16', 'n18')), ('n4', 'n12'), ('n5', 'n13'), ('n6', 'n15'), ('n7', 'n19')

Table 5.13: vertex mappings given by GT, completeMMAS, and reducedMMAS when pairing model 032 with its noise levels in Fig 5.17. The difference between GT mapping and completeMMAS is in red; the difference between GT mapping and reducedMMAS is in blue.

5.3 Comparison ACO with the greedy and random search for ExGED problem

ACO belongs to meta-heuristics, and it is a global-search method [Dorigo and Stützle, 2019]. Therefore, it may avoid local solutions better than other meta-heuristic approaches. To verify that, we compare the ACO-based proposed approach with other meta-heuristic approaches for solving the ExGED problem. Specifically, the greedy [Black, 2005] and random search [Rastrigin, 1963] algorithms are utilized for the comparison.

Greedy search The greedy algorithm constructs a solution only based on the heuristic information. It begins with an empty edit path. Then, it adds a new candidate to the current edit path such that this candidate has the lowest cost among other candidates. The step is repeated until the edit path is complete. Although the solution produced by greedy search is deterministic, it has a high chance to be stuck in a local solution. Therefore, we also integrate LS with the algorithm to improve the solution quality. The LS used is 3-opt as for ACO. So, we compare ACO with greedy search (called Greedy) and enhanced greedy search (called Greedy-LS).

Random search The random search algorithm constructs a solution in a stochastic manner. It starts with an empty edit path. At each iteration, a random index of all possible candidates/vertex mappings is generated following a uniform distribution. Then,

a candidate/vertex mapping corresponding to this index is selected to append into the current edit path. Similar steps are done until the current edit path is complete. Hence, the solution yielded by random search is non-deterministic. It might obtain either a local solution or a global solution regarding the generated indices. Likewise the Greedy, we combine the random search with the LS to enhance the solution. Therefore, we also compare ACO with random search (called Random) and random search with LS (called Random-LS).

With Greedy, only heuristic information is used to find the solution. With Random, only stochastic factor is utilized for the solution construction, but ACO combines heuristic and pheromone factors to establish an edit path. Comparing ACO with these approaches helps us to clarify the following points. Compared with the greedy algorithm, we would like to know whether or not the pheromone factor in ACO is essential, i.e., whether or not it will contribute significantly to the solution quality as the heuristic factor does. Comparing ACO with the random search for the ExGED problem will verify whether the dataset is too easy to be solved by Random, and so ACO is not necessary. Furthermore, it is possible whether LS helps Greedy and Random reach as competitive solutions as ACO.

To give an answer to these questions, we design the experiments on two aspects: GM and classification problems. Datasets used for these problems are as follows. For the GM problem, the dataset B with GT in Appendix A is used to verify the low-level matching, i.e., the vertex-to-vertex mappings and the edge-to-edge mappings. The edge-to-edge mappings can be implied from the vertex-to-vertex mappings, and we show only the results for the vertex-to-vertex mappings. For the classification problem, the dataset C is used. We take 20 models as a train set. In each model, 30 noisy symbols are selected as a test set. All of them are chosen randomly. We have 20 classes in total. For all experiments in this section, we use improved MMAS for solving the ExGED problem. We call MMAS-ExGED for short. The classifier is K-NN with $K = 1$ [Fix and Hodges, 1989]. We evaluate the results of the GM problem in terms of ExGED distance and computational time. We analyze the results of the classification problem according to classification rate and classification time. Moreover, we investigate more thoroughly the classification results via the confusion matrix.

5.3.1 Comparison in the GM context

Fig. 5.18 (page 134) shows the average results of all compared methods in the GM context. Here, one needs to pair the models with their noisy symbols at several levels. To be fair, we set $it_{\max} = 300$ for the Random. That means it will repeat the solution building process 300 times, and it will keep the best solution with the smallest ExGED distance at the end. Due to this setting, the computational time of Random is higher than Greedy but still lower than MMAS. Besides, there is no significant difference between Greedy, Random and their versions with LS. About the ExGED distance, MMAS gives the lowest (best) ExGED distance among the five compared solvers.

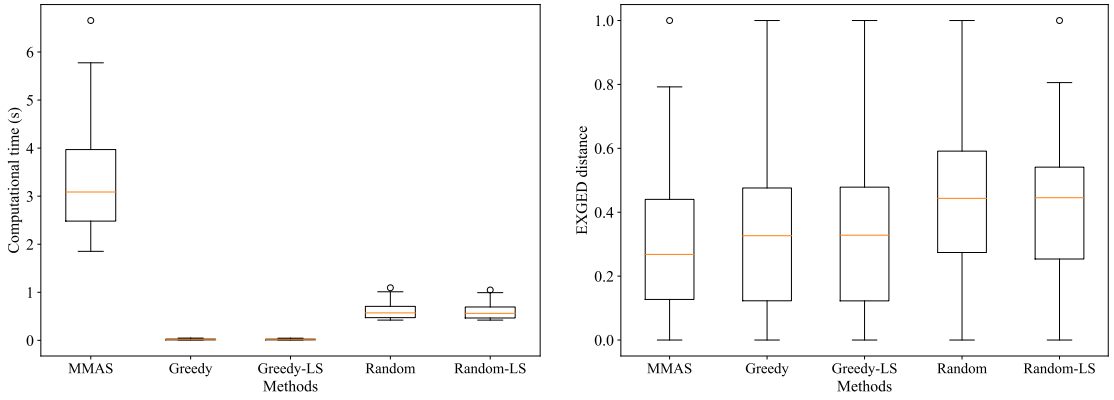


Figure 5.18: Comparison MMAS with greedy and random search in GM context for symbol dataset in terms of computational time (left) and ExGED distance (right).

5.3.2 Comparison in the classification context

Table 5.14 (page 135) presents results for the classification problem. We examine with $k = \{2, 3\}$ and $it_{\max} = 300$. 'Accuracy' is the percentage of the number of testing symbols correctly classified. $\#matchings$ is the total number of matchings done for each method. For Greedy and Random, $\#matchings = \#test \times \#train$. For MMAS-ExGED, $\#matchings = \#test \times \#train \times pop \times it_{\max}$. 'Time/matching(s)' is the ratio between the total computational time and the total number of matchings.

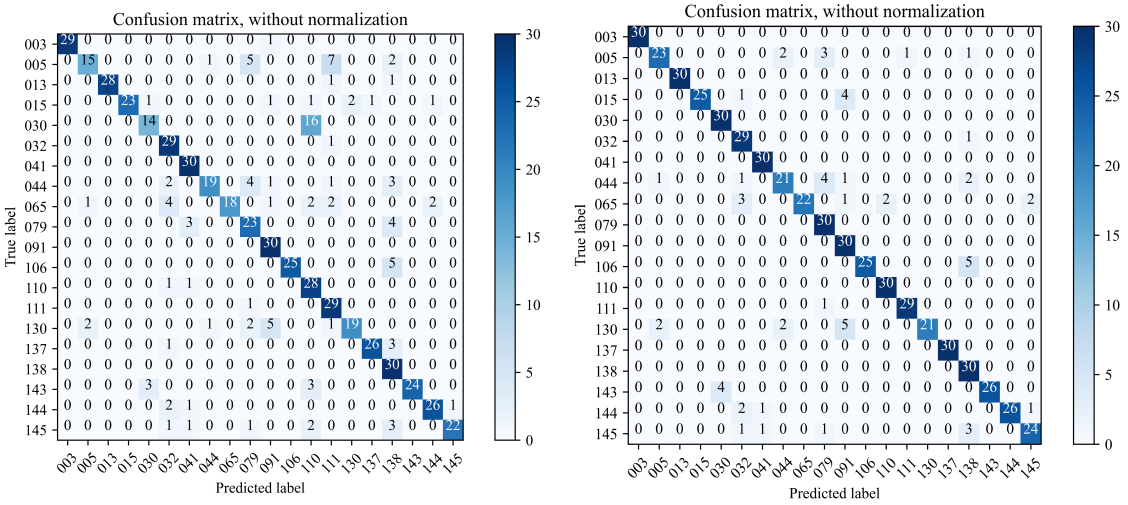
The results show no remarkable improvements for the Greedy and Random with different values of k. Moreover, the LS does not help the Greedy method to achieve a better classification rate. In contrast, Random obtains higher accuracy when integrating with the LS. It witnesses an increase from around 60% to 83.83% with LS. Nevertheless, its results are not stable because of the stochastic scheme. The accuracy of 83.83% is the best one it can reach with $k = \{2, 3\}$ and $it_{\max} = \{50, 100, 200, 300\}$. MMAS provides the most robust results for classifying noisy symbols. With both values of k, it gains approximately 90% accuracy. Its complex mechanic to construct a solution can help to achieve these results in comparison with Greedy and Random methods. Although MMAS is more robust regarding accuracy, it is relatively more time-consuming than Greedy and Random. A parallel version of MMAS (ACO in general) can be useful to reduce the computational time.

We investigate the confusion matrices from the classification results derived by the solvers to understand the results. We select the best versions of Greedy and Random that provide the highest classification rates to analyze. They are greedy without LS (Greedy) and random search with LS (Random-LS). Along with MMAS with LS, three confusion matrices are given in Fig. 5.19 and Fig. 5.20 in pages 135 and 136, respectively. We found that the classes which have frequently misclassified noisy symbols are 005, 030, 079, and 130. For class 005, all three methods (MMAS, greedy, random search) classify incorrectly into class 079 with the highest frequency. For class 030, all its noisy symbols are correctly categorized with MMAS, 16 of them with Greedy, and 24 of them with Random-LS. A similar observation is for class 079. For class 130, the testing symbols are regularly arranged

Solver	$(k, it_{\max})$	Accuracy (%)	Time/matching (s)	Total time (s)	*#matchings*
Greedy	(2,_)	81.17	0.19	115.90	600
	(3,_)	81.17	0.20	121.76	600
Greedy-LS	(2,_)	80.33	0.20	117.27	600
	(3,_)	80.33	0.21	125.76	600
Random	(2, 300)	68.50	0.03	4562.51	180000
	(3,300)	66.00	0.02	4498.54	180000
Random-LS	(2, 300)	83.83	0.03	5690.65	180000
	(3,300)	83.83	0.03	5689.65	180000
MMAS-LS	(2,300)	89.93	0.03	30754.33	900000
	(3,300)	90.17	0.04	32279.08	900000

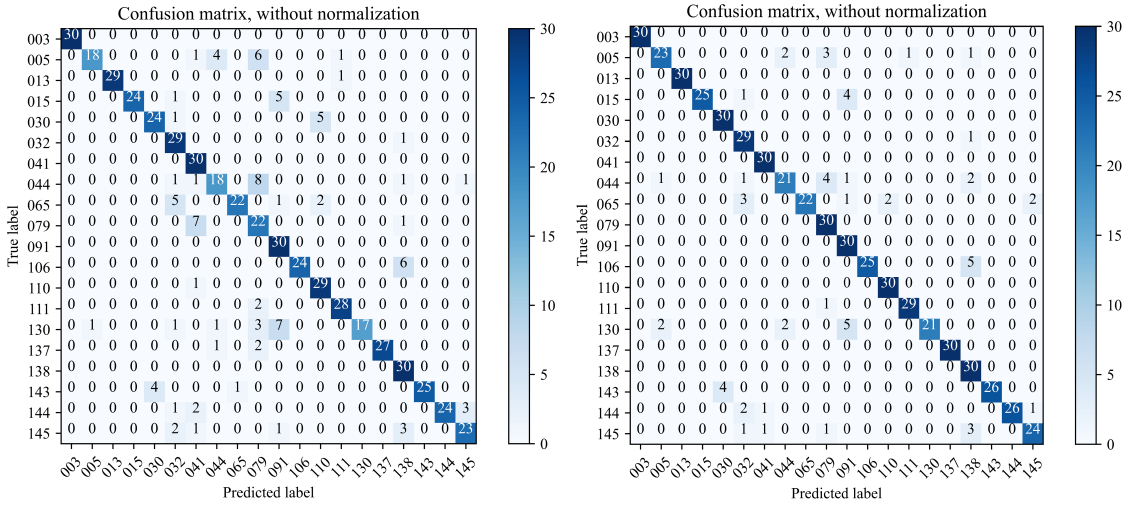
Table 5.14: Symbol classification results given by MMAS, greedy search, and random search. Time/matching (s) is the fraction betweent the total time and the number of matchings. The highest accuracy result is in red.

into class 091 for three methods. Overall, these classes have a lot of common parts with other classes such as 079, 110 (Fig. 5.21 (page 137)). Therefore, it is more difficult. Despite that, MMAS gives the highest classification rate among the three methods. These investigations with the general classification results demonstrate a robust performance of ACO, MMAS in particular.



(i) Greedy (ii) MMAS

Figure 5.19: Comparison of confusion matrices from the classification results of Greedy and MMAS on the symbol dataset.



(i) Random (ii) MMAS

Figure 5.20: Comparison of confusion matrices from the classification results of Random Search and MMAS on the symbol dataset.

5.4 Evaluation between multivalent and univalent GM on symbol dataset

As mentioned in chapter 2, multivalent GM is more general than univalent matching. That is because multivalent matching can cover more situations than one-to-one correspondence. We verify here by comparing GED (univalent) and ExGED (multivalent) on a real dataset. The dataset contains distorted symbols being deformed, rotated, or scaled as introduced in subsection 5.2.1. As a result, the attributes of these symbols and their structures are modified. So, it is questionable that:

- whether or not GED and ExGED can have good performances on such noisy dataset;
- whether or not it is necessary to have splitting and merging operations in ExGED.

To validate these points, we evaluate GED and ExGED in two contexts: GM problem and classification problem.

The datasets used for GM context and classification context are B and C, respectively. The classifier is also K-NN with $K = 1$. The evaluation criteria are the same as in the section 5.2.2.5. For solving the GED problem, we employ the bipartite graph matching of [Riesen et al., 2007b], called BP-GED. More detail of this solver can refer to the State-of-the-art in subsection 3.1.4. The common cost functions, such as substitution, deletion, insertions for vertices and edges, are similar for GED and ExGED. All the cost functions can be referred to Table 5.5 (page 116).

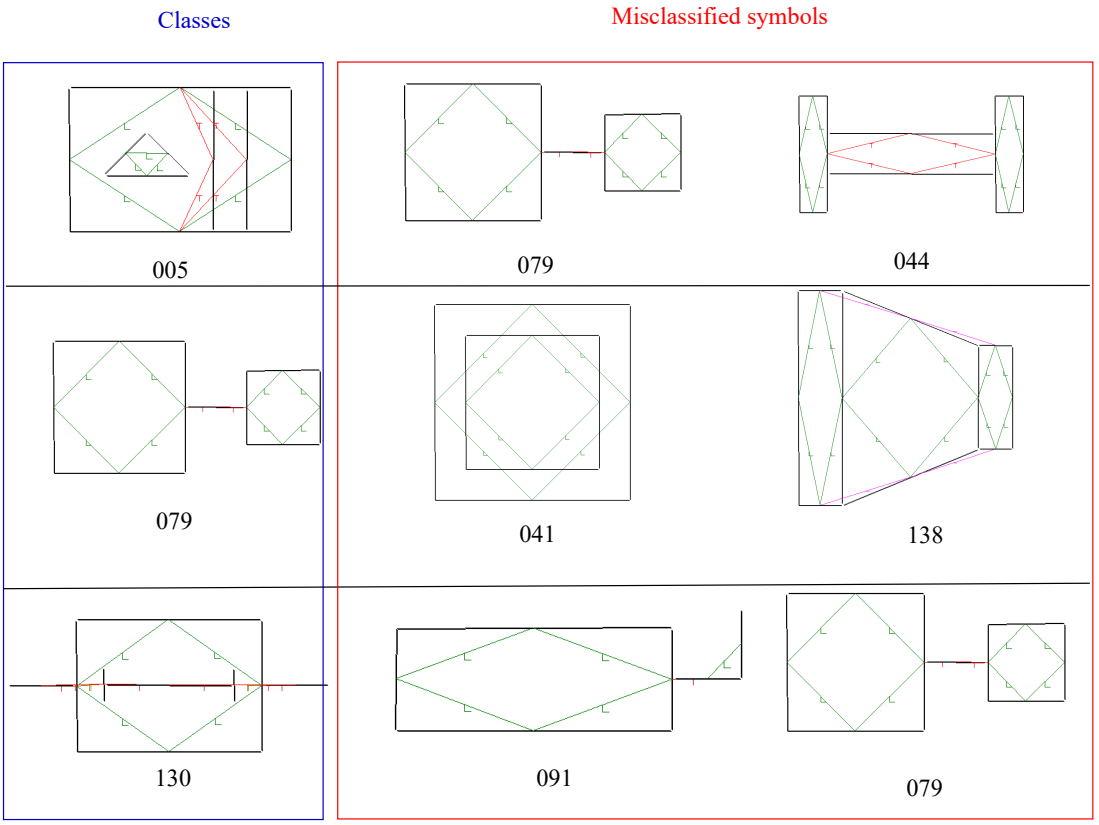


Figure 5.21: Draw of vertices and edges on the classes (left) that the noisy symbols are usually misclassified into wrong classes (right). Notice that these classes have similar graph representations with L-type edges, which lead to confusions and errors during the classification process.

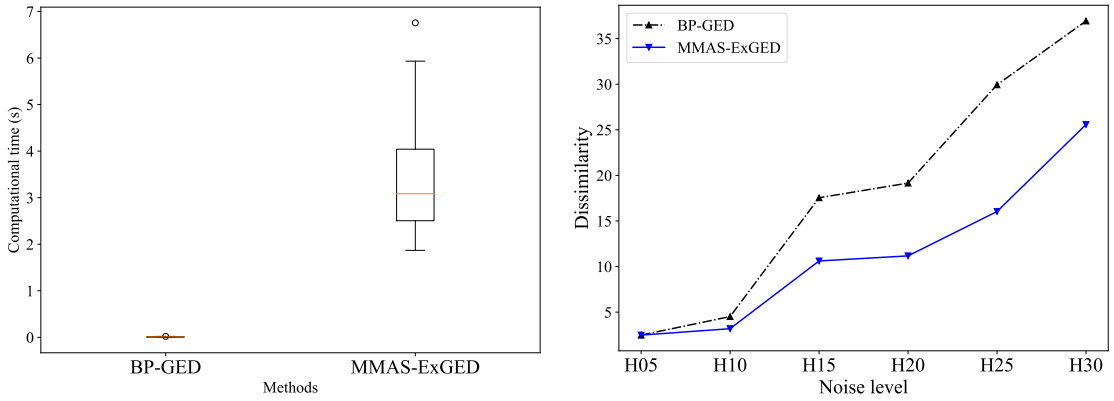
5.4.1 Comparison in the GM context

Fig. 5.22 (page 138) presents results for the GM problem. It notices that the dissimilarity increases in accordance with the noise levels (Fig. 5.22ii (page 138)). At the lowest noise level, H05, both BP-GED and MMAS-ExGED yield the same dissimilarity measure. It can imply that there is no need to use splitting and merging operations for this noise level. Only classical GED operations are robust enough to do matching. Maybe this confirmation is not held for higher noise levels. We investigate it through the low-level matching produced by the two methods. It shows that MMAS-ExGED achieves more meaningful mappings than BP-GED at distorted positions.

We give an example in Fig. 5.23 (page 139) and Table. 5.15 (page 139) for model 032. As shown in Table 5.15, BP-GED has more dissimilar mappings when the noise level grows up. In contrast, MMAS-ExGED's mappings resemble more to GT. At a noise level of 05, both approaches give the same mappings with the GT mappings. At noise level of 10, BP-

GED decides to substitute $n1 \rightarrow n5$, and insert $\varepsilon \rightarrow n9$. However, MMAS-ExGED decides to split $n1 \rightarrow \{n5, n9\}$, which helps MMAS-ExGED to achieve the matching equivalent with GT thanks to primary edit operations in GED and its extended operations: splitting and merging. Moreover, the mapping is closer to the real status of the symbol (its line is broken into two lines). A similar case is observable for noise level of H15 in Table 5.15 (page 139). Hence, it gives more meaningful and explainable mapping than the traditional GED.

Nevertheless, MMAS-ExGED is much slower than BP-GED in terms of computational time (Fig. 5.22i). It is due to these following reasons. First, the cost matrix for GED has smaller size than the one for ExGED. Secondly, BP utilizes Hungarian method for solving the GED with the complexity of $\mathcal{O}((m+n)^3)$, where m, n are the number of vertices in G and H, respectively. Moreover, BP is a deterministic algorithm and it gives only one matching/solution at the end. MMAS solves ExGED in an iterative way. For one ant, it can requires $\max(m+h+1, n+l+1)$ steps to construct a complete matching, where h, l are sizes of merging and splitting sets, respectively. And it is repeated for *pop* ants during $it_{\max}$ iterations. Furthermore, the pheromone update process also takes $\mathcal{O}((m+h+1)\times(n+l+1))$ at each iteration, excluding the pheromone adjustment process. The LS also adds an extra time to MMAS-ExGED. In total, at one runtime, it would take more time than BP-GED to give the best solution at the end.



(i) Computational time (s)

(ii) Dissimilarity by noise levels

Figure 5.22: Comparison between GED-based methods and ExGED-based method for symbol dataset in terms of computational time and dissimilarity.

5.4.2 Comparison in the classification context

Through Table 5.16, we observe the following main points. Firstly, the ExGED gives a better accuracy but is more time-consuming than the BP-GED. Nevertheless, constructing one matching of one ant in MMAS-ExGED is less time-consuming. Secondly, between GED-based and ExGED-based approaches, the ExGED-based ones are usually more robust with a higher classification rate. Thirdly, better results are obtained with $k = 3$ rather than $k = 2$ since it provides extra matching possibilities. In contrast, it demands more

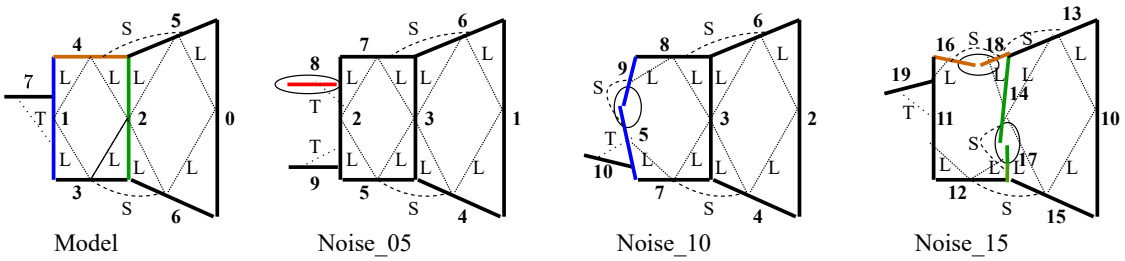


Figure 5.23: Drawing of vertices (in bold) and edges (dashed lines) on the model 032 and its noise levels. Ellipses indicate noise positions, same noise position illustrated with the same color.

	032_H05	032_H10	032_H15
GT	('n0', 'n1'), ('n1', 'n2'), ('n2', 'n3'), ('n3', 'n5'), ('n4', 'n7'), ('n5', 'n6'), ('n6', 'n4'), ('n7', 'n8'), ('eps', 'n9')	('n0', 'n2'), ('n1', ('n5', 'n9')), ('n2', 'n3'), ('n3', 'n7'), ('n4', 'n8'), ('n5', 'n6'), ('n6', 'n4'), ('n7', 'n10')	('n0', 'n10'), ('n1', 'n11'), ('n2', ('n14', 'n17')), ('n3', 'n12'), ('n4', ('n16', 'n18')), ('n5', 'n13'), ('n6', 'n15'), ('n7', 'n19')
BP	('n0', 'n1'), ('n1', 'n2'), ('n2', 'n3'), ('n3', 'n5'), ('n4', 'n7'), ('n5', 'n6'), ('n6', 'n4'), ('n7', 'n8'), ('eps', 'n9')	('n0', 'n2'), ('n1', 'n5'), ('n2', 'n3'), ('n3', 'n7'), ('n4', 'n8'), ('n5', 'n6'), ('n6', 'n4'), ('n7', 'n10'), ('eps', 'n9')	('n0', 'n16'), ('n1', 'n11'), ('n2', 'n14'), ('n3', 'n12'), ('n4', 'n17'), ('n5', 'n13'), ('n6', 'n18'), ('n7', 'n19'), ('eps', 'n10'), ('eps', 'n15')
MMAS-ExGED	('n0', 'n1'), ('n1', 'n2'), ('n2', 'n3'), ('n3', 'n5'), ('n4', 'n7'), ('n5', 'n6'), ('n6', 'n4'), ('n7', 'n8'), ('eps', 'n9')	('n0', 'n2'), ('n1', ('n5', 'n9')), ('n2', 'n3'), ('n3', 'n7'), ('n4', 'n8'), ('n5', 'n6'), ('n6', 'n4'), ('n7', 'n10')	('n0', 'n10'), ('n1', 'n11'), ('n2', ('n14', 'n17')), ('n3', 'n12'), ('n4', ('n16', 'n18')), ('n5', 'n13'), ('n6', 'n15'), ('n7', 'n19')

Table 5.15: Compare vertex mapping given by BP-GED and MMAS-ExGED with the GT mapping for model 032, parts differing from those of GT in red color.

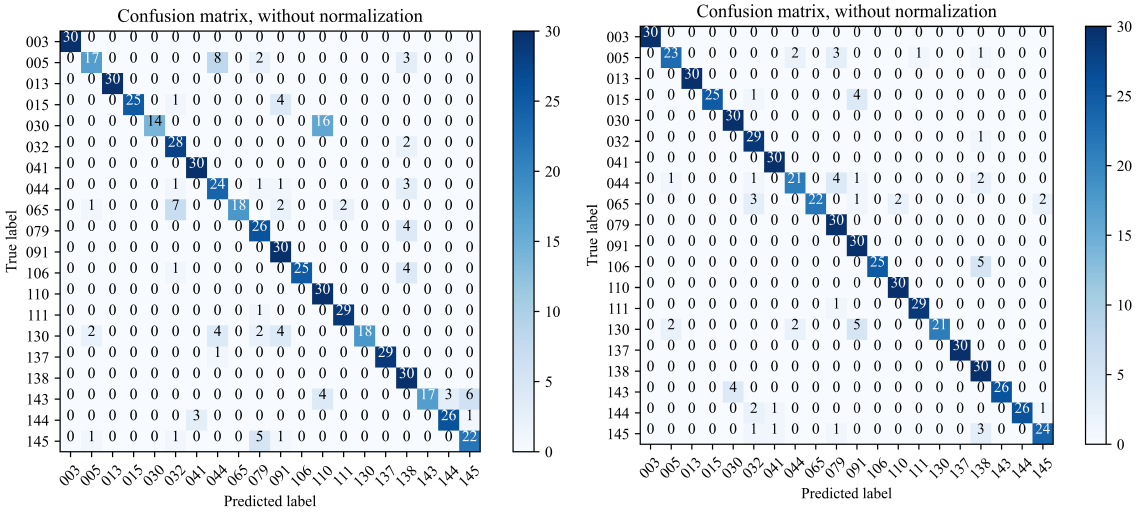
computational time due to the size increase of the cost matrix. Finally, $(k, it_{\max}) = (3, 200)$ can bring a satisfied result for the classification problem.

Solver	$(k, it_{\max})$	Accuracy(%)	Time/matching(s)	Total time(s)	$\#matchings$
BP-GED		83.00	0.13	75.399	600
MMAS-ExGED	(2,50)	88.50	0.03	5151.49	150000
	(3,50)	89.33	0.03	5188.16	150000
	(2,100)	89.33	0.03	10204.31	300000
	(3,100)	89.33	0.03	10276.15	300000
	(2,200)	89.67	0.03	20905.48	600000
	(3,200)	90.17	0.04	21132.08	600000
	(2,300)	89.83	0.03	30754.33	900000
	(3,300)	90.17	0.04	32279.08	900000

Table 5.16: Symbol classification results given by BP-GED and MMAS-ExGED, the highest accuracy result is in red.

We investigate more deeply the classification results of BP-GED and MMAS-ExGED through their confusion matrices in Fig. 5.24 (page 140). In Fig. 5.24i, the most misclassified symbols belong to class 030, with more than 50% of the wrong classification into class 110. Symbols of classes 005 and 065 have almost the same misclassified rates. The MMAS-ExGED's confusion matrix is more fruitful. In contrast to BP-GED's result for class 030, MMAS-ExGED classifies all of 30 testing symbols correctly. Classes 005 and 065 witness some mis-classifications, but with lower rates. We enumerate some misclassified symbols in the class of 030, the model 030, and model 110 in Fig. 5.25 (page 140). We observe that the graph representation of model 030 is somehow different from one of its

noisy symbols. The one from model 110 has a lot of similar parts. That can partly explain the wrong results of BP-GED. MMAS-ExGED gets a superior result thanks to splitting and merging operations.



(i) BP-GED (ii) MMAS-ExGED

Figure 5.24: Confusion matrices from the classification results of BP-GED and MMAS-ExGED for symbol dataset. Values on the matrix diagonal are the true positives. For each class, there are 30 testing symbols.

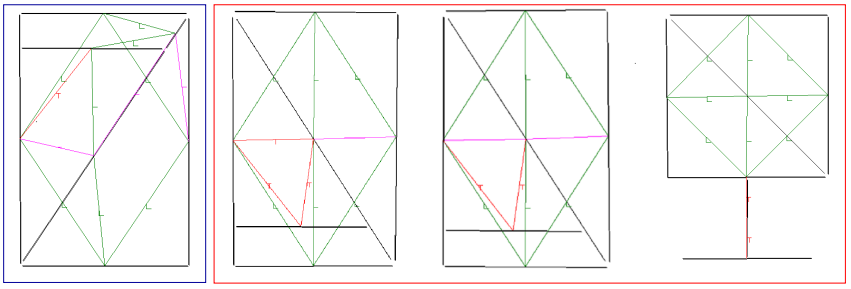


Figure 5.25: Draw of vertices and edges on the model 030, its testing symbols, and the model 110. It seems that the structures of noisy symbols in class 030 are more similar to that of the class 110, which leads to the misclassification.

5.5 Summary

In this section, we have proposed an Ant Colony Optimization for the EXGED, called ACO-ExGED. The ACO is selected due to its advantages for the combinatorial problems. First, a general pipeline for ACO-ExGED is designed. A construction graph G_{ants} is

built to be the search space for the ants. Then, two searching strategies are introduced for ACO-ExGED. The first one considers all the remaining candidates. Meanwhile, the second strategy only considers the neighborhood of the previously selected vertices. As a result, the second strategy is more time-saving, but it can miss good candidates. These two strategies aim to study the trade-off between solution quality and computation time. Next, the local searches are applied to enhance the solution quality obtained by ACO. Finally, the algorithm outputs the matching (edit path) and the corresponding dissimilarity (ExGED distance).

After proposing ACO-ExGED, experiments are designed to evaluate the algorithm. Due to the lack of dataset, we have built a symbol dataset for the evaluation. The dataset is accessible online for those interested in multivalued GM. We evaluate ACO-ExGED under several scenarios. First, hyper-parameter tuning for ACO is required in order to find a suitable set of parameters for the dataset. A comparison between three variants of ACO is also implemented. It shows that MMAS obtains superior performance than AS and ACS for the symbol dataset. Considering two searching strategies, a better trade-off between the computational time and the solution quality is observed for the neighborhood search. In addition, we also study the importance of the heuristic and stochastic factors of ACO by comparing with Greedy and Random Search. It shows that ACO achieves higher accuracy thanks to these factors. However, ACO is more time-consuming than the Greedy and Random Search. Last but not least, an evaluation between the univalent GM and multivalent GM is figured out. Thanks to the splitting and merging operations, ExGED gives better matchings which are more meaningful and explainable than ones produced by univalent GM, BP-GED in this case. That is true both in GM and classification contexts. Nevertheless, ACO-EXGED needs more improvements to reduce its computational time for the classification task.

Part III

Conclusion

Chapter 6

Conclusion and Future works

Contents

6.1 Conclusion 144
6.2 Perspectives and future challenges 147

Abstract
In this last chapter, we remind our main studied topic as well as the principal contributions. Also, some perspectives are suggested in the upcoming time.

6.1 Conclusion

This thesis is dedicated to structural data representation and analysis. The comparison between objects is achieved by resolving a problem of Graph Matching (GM), where similarity is evaluated regarding vertex and edge mappings. Multivalent GM can cover more situations in real-life contexts thanks to its extensions: many-to-one and one-to-many mappings. To the best of our knowledge, studies for multivalent GM are still very limited compared with univalent GM. Motivated by this fact, we concentrate on the multivalent GM in both theoretical and practical aspects.

The contributions of this thesis are presented in two chapters. Chapter 4 is dedicated to the theoretical aspect of multivalent graph matching and Chapter 5 describes the proposed solver we defined along with its empirical evaluations.

In Chapter 4, we propose a formal definition for the multivalent GM in the form of an extended Graph Edit Distance (GED). There exists several works addressing the multivalent GM as an extension of GED because of the flexibility and adaptability of this method. Nonetheless, no formal definition is given. We have reformulated the multivalent GM as an extension of GED by adding splitting and merging edit operation on vertices and edges. Based on the proposed formulation, a comprehensive analysis is done for two key factors influencing ExGED's complexity and efficiency. The first is parameter k, which defines the maximum number of vertices one vertex can match. The higher k is, the more combinations of vertices will become possible during the matching process. This combinatorial explosion is illustrated through numerical simulations demonstrating the need to

carefully select the value of k according to the application domain. The second analysis is concerned with the edge transformations induced by the vertex splitting/merging. In the literature review, most related works ignore or mention this problem very shortly. The mapping between edges is vital in GM because edges describe the structural information inside graphs. In multivalent GM, vertices have more matching possibilities, so the edges are also strongly impacted. Indeed, by enumerating all possible cases, we demonstrate that the edge relations induced by vertex mappings in ExGED are far more complicated than ones in univalent matching. To the best of our knowledge, neither GED nor other formulations of multivalent matching discussed in details the induced operations on edges. Therefore, this analysis provides an insight into ExGED, which helps to design better solvers for ExGED and multivalent GM overall.

In Chapter 5, we propose a meta-heuristic approach for solving the ExGED problem. Particularly, Ant Colony Optimization (ACO) is selected to tackle the high complexity thanks to the capacity of ACO to solve efficiently combinatorial problems such as quadratic assignment problem. This is the first time an ACO-based approach is used to deal with a GED-based multivalent GM. The performances are evaluated based on the dissimilarity measures between graphs (ExGED distances) which are derived from the selected edit paths. These edit paths consist of vertex/edge mappings from two compared graphs such that only one operation is applied on each vertex/edge of both graphs. Inheriting the GED's property, the proposed approach can deal with graphs having numeric and symbolic labels. Another remarkable point is that the edit cost matrix is provided as an heuristic information for ACO. This matrix presents the goodness of vertex mappings as in [Riesen et al., 2007b]. We also integrate the estimated edge costs for each vertex mapping thanks to the analysis done in the Chapter 4. This cost matrix helps ACO to search for the best solution, to save the memory, and to reduce the computational time because all heuristic information is stored at one time and can be used by all ants. We denote the proposed approach as ACO-ExGED.

In this thesis, two versions of ACO-ExGED are presented. The first version explores the complete search space and the second version tries to explore a reduced search space by applying a neighborhood search. This technique relies on the neighbors of the previously selected vertex in the source graph to decide the following candidates. If the previously selected vertex has no neighbor, a random candidate will be chosen. Therefore, the second version is less time-consuming than the first version, especially for sparse graphs. The local searches are also coupled with two versions to enhance its solution quality. These different versions of ACO-ExGED relate to the question of the trade-off between the computational time and the solution quality, which is discussed in detail in the experimental section.

Due to the lack of datasets dedicated to multivalent GM, we generate one from a public symbol dataset called SESYD. Graph representations are generated from SESYD initial and distorted images (with various types of noise). Therefore, their graph representations differ both in labels and structure. A subset of pairs of graphs with the corresponding matching Ground Truth (GT) has also been included to serve for a possible performance evaluation at the graph-matching-level. Multiple levels of noise ranging from low to high distortions are considered in this subset. The dataset is freely accessible for those who wish to verify their Graph Matching methods (univalent or multivalent as well). The target is to facilitate the evaluation process of multivalent GM, and it can be seen as another modest

contribution.

The last contribution is in Chapter 5 to study the behaviors of ACO meta-heuristic for solving the ExGED problem. Additionally, several experiments are conducted under different aspects. Almost all the evaluations are performed on the aforementioned symbol dataset due to lack of dataset. First, a comparison between variants of ACO is carried out in the GM context. It shows that Max-Min Ant System (MMAS) and Ant Colony System (ACS) are rather competitive and they obtain superior performances than Ant System (AS) in terms of accuracy and computational time. However, the analysis at the matching level demonstrate that MMAS provide results relatively closer to the GT than those of ACS. From these results, one can say that MMAS is more suitable for solving the ExGED problem on the symbol dataset, and it is utilized for other evaluations afterwards.

In addition, two versions of ACO-ExGED are evaluated in the GM context. The results are also examined quantitatively (by looking to the provided ExGED distances and computational times) and qualitatively (by looking to the provided matchings compared to GT). We notice that the ExGED distances yielded by the two versions are pretty similar and closed to GT distance. However, ACO-ExGED with neighborhood search spends much less time to compute the final solution. That confirms the efficiency of the neighborhood search to reduce the search space, which leads to less time-consuming. The matching quality verification compared with the GT mappings shows that the ACO-ExGED with the complete search space gives more precise matching generally, i.e., the matching are more similar to the GT. However, both versions of ACO-ExGED give relatively similar mappings for symbols at low noise levels. These results answer the question of the trade-off between accuracy and speed. Neighborhood search can help ACO-ExGED to accelerate the computational time but generate less qualitative solutions than the ACO-ExGED with complete search space. For the symbol dataset, this trade-off is acceptable, so we utilize ACO-ExGED with neighborhood search for later experiments, called ACO-ExGED for short.

After studying the proposed method with its variants and different strategies for building a solution, another evaluation is brought out with other meta-heuristics. The greedy and the random search are used to justify the usefullness of the heuristic information and the random scheme in ACO. The results obtained in a Graph Matching task and in a classification task show that ACO is superior to greedy and random search with respect to accuracy. Nevertheless, ACO spends more computational time than greedy and random search to provide the solutions. For the classification problem, the accuracy of ACO with Local Search (LS) is higher than 90%, while the best results for greedy and random search algorithms coupled with the LS are around 81% and 83%, respectively. Without LS, the random search algorithm cannot reach 70% accuracy over 300 iterations.

The last experiment shows the usefulness of extended edit operations in ExGED in comparison with the classical operations in GED. A comparison with Bipartite Matching in [Riesen et al., 2007b], called BP-GED, reveals that the extended operations (splitting and merging) help ExGED to achieve better classification results on the symbol dataset. The analysis at the matching level shows more explainable and meaningful results than those of GED. Matchings from ExGED can reflect the real status of the symbols thanks to the additional edit operations of splitting and merging. For example, splitting one vertex

into a set of vertices represents the relation between the original line in the symbol and its broken lines in the noisy one, and that is impossible for GED. So, the generality and utility of ExGED are partly demonstrated via this experiment. Furthermore, ExGED has a better classification result than GED for all given values of k and the number of iterations $it_{\max}$. These results highlight the usefulness of splitting and merging operations in ExGED.

6.2 Perspectives and future challenges

This thesis focuses on solving the Multivalent Graph Matching and proposes to use ACO meta-heuristic to solve this problem reformulated as an extended graph edit distance. Even if theoretical and practical aspects have been studied through the thesis, there is still room for improvements.

In this work, we formulate the multivalent GM as an extended GED and analyze the edges mapping between two graphs only as a consequence of the vertex operations (splitting/merging). The analysis shows the complexity of these edge relations. Therefore, we implement it in a simplified way using Munkrees algorithm. However, one can try to consider the edge mapping more precisely during the edit path construction to realize an exact multivalent GM resulting in better accuracy.

In addition, one weak point of the GED-based approach is the need to define the cost functions for each edit operation. Determining cost functions for classical operations in GED is not a trivial task, even harder for splitting/merging operations in ExGED. Also, the cost functions regularly depend on the dataset. In the thesis, the manual way is used. Nevertheless, one can try to learn all cost functions in ExGED automatically, which makes the proposed methods more autonomous.

ACO has been selected for the resolution of ExGED. Classical ACO's scheme with basic mechanisms is used. Maybe more sophisticated mechanisms will provide better solutions. Besides, we test only three principal ACO variants (AS, MMAS, ACS). So, other ACO variants can be applied to the ExGED. Moreover, ACO is known to be easily converted into parallel computing, which can help to accelerate the computation process. So, distributed version of ACO will take less time than the recently proposed methods in the thesis. Lastly, more meta-heuristics, such as genetic algorithm, simulated annealing, particle swarm optimization, should be compared with ACO to have more general evaluations.

Another perspective is related to the evaluation of the multivalent GM. Currently, the Benchmark datasets for multivalent GM evaluation are very limited. Therefore, it is vital to have more available datasets for evaluating and justifying the implemented solvers. In addition, the graphs generated from the dataset we use to test the proposed methods are relatively small. It would be better to have larger graphs in various scenarios for a more general evaluation of the proposed methods.

From a multivalent matching point of view, we have not compared the proposed methods with other approaches that able to tackle multivalent or many-to-many matching. For example, mewthods which are based on continuous relaxation [**?**], beam-search [Ambauen et al., 2003], or graph embedding [Demirci et al., 2006]. It might bring interesting results when comparing them together. Recently, deep learning is also an ongoing trend for per-

forming the univalent GM problem and it would be possible for the multivalent GM. It could be a challenging task but might bring fruitful results.

Last but not least, it would be interesting to discuss the anytime property of the proposed methods. Inspired from the thesis of [Abu-Aisheh, 2016], the GM problem can be converted into anytime by introducing stopping constraints associated to time or to memory consumption. ACO improves the solution quality over iterations. A longer time with more iterations will help ACO to achieve qualitative solutions except for the case of convergence. Therefore, ACO for ExGED can become an anytime matching by providing the solution at the requested moment according to the given constraint. The anytime GM turns the GM problem to be more flexible to the user's demands.

Appendices

Appendix A

Ground truth of vertex-to-vertex correspondences for a subset in the symbol dataset

002_H05	**002_H10**	**002_H15**	**002_H20**	**002_H25**	**002_H30**
$0 \to 6$	$0 \to 6$	$0 \to 8$	$0 \to 6$	$0 \to \{13, 16\}$	$0 \to \{11, 15, 19, 21\}$
$1 \to 7$	$1 \to 7$	$1 \to 7$	$1 \to \{7, 13\}$	$1 \to \{15, 18\}$	$1 \to 10$
$2 \to 8$	$2 \to 9$	$2 \to 10$	$2 \to 9$	$2 \to \{17, 22\}$	$2 \to \{13, 17, 18\}$
$3 \to 9$	$3 \to 8$	$3 \to 9$	$3 \to 8$	$3 \to 12$	$3 \to \{12, 23\}$
$4 \to 11$	$4 \to 11$	$4 \to \{12, 15\}$	$4 \to \{12, 14\}$	$4 \to \{20, 21\}$	$4 \to 14$
$5 \to 10$	$5 \to 10$	$5 \to \{11, 16\}$	$5 \to 10$	$5 \to 14$	$5 \to \{16, 24\}$
$6 \to 13$	$6 \to 12$	$6 \to 13$	$6 \to 15$	$6 \to 19$	$6 \to 20$
$7 \to 12$	$7 \to 13$	$7 \to 14$	$7 \to 11$	$7 \to 23$	$7 \to 22$
032_H05	**032_H10**	**032_H15**	**032_H20**	**032_H25**	**032_H30**
$0 \to 1$	$0 \to 2$	$0 \to 10$	$0 \to 15$	$0 \to \{11, 14, 16, 20\}$	$0 \to \{11, 13\}$
$1 \to 2$	$1 \to \{5, 9\}$	$1 \to 11$	$1 \to \{19, 20, 23\}$	$1 \to \{8, 19\}$	$1 \to \{14, 22, 25\}$
$2 \to 3$	$2 \to 3$	$2 \to \{14, 17\}$	$2 \to 13$	$2 \to \{9, 18\}$	$2 \to \{15, 17\}$
$3 \to 5$	$3 \to 7$	$3 \to 12$	$3 \to \{17, 22\}$	$3 \to 10$	$3 \to 12$
$4 \to 7$	$4 \to 8$	$4 \to \{16, 18\}$	$4 \to \{16, 21\}$	$4 \to \{15, 17, 23\}$	$4 \to \{18, 20, 23\}$
$5 \to 6$	$5 \to 6$	$5 \to 13$	$5 \to 14$	$5 \to \{13, 21\}$	$5 \to \{19, 24, 26\}$
$6 \to 4$	$6 \to 4$	$6 \to 15$	$6 \to 18$	$6 \to \{12, 22\}$	$6 \to \{16, 21\}$
$7 \to 8$	$7 \to 10$	$7 \to 19$	$7 \to \varepsilon$	$7 \to 24$	$7 \to \varepsilon$
$\varepsilon \to 9$					
064_H05	**064_H10**	**064_H15**	**064_H20**	**064_H25**	**064_H30**
$2 \to 1$	$2 \to 7$	$2 \to 9$	$2 \to 10$	$2 \to \{10, 15\}$	$2 \to \{18, 28, 33\}$
$3 \to 2$	$3 \to 8$	$3 \to 8$	$3 \to 11$	$3 \to 9$	$3 \to \{20, 22, 27, 31\}$
$4 \to 3$	$4 \to 10$	$4 \to 12$	$4 \to 12$	$4 \to \{18, 20\}$	$4 \to 19$
$5 \to 4$	$5 \to 9$	$5 \to \{15, 17\}$	$5 \to 13$	$5 \to 17$	$5 \to \{26, 29\}$
$6 \to 8$	$6 \to 14$	$6 \to 11$	$6 \to \{17, 20\}$	$6 \to 14$	$6 \to 21$
$7 \to 7$	$7 \to 11$	$7 \to 13$	$7 \to 16$	$7 \to 11$	$7 \to 32$
$8 \to 5$	$8 \to 12$	$8 \to \{14, 19\}$	$8 \to 14$	$8 \to 12$	$8 \to 34$
$9 \to 6$	$9 \to 13$	$9 \to 10$	$9 \to 15$	$9 \to 13$	$9 \to \{23, 30\}$
$10 \to 9$	$10 \to 15$	$10 \to 18$	$10 \to 18$	$10 \to 19$	$10 \to 24$
$11 \to 10$	$11 \to 16$	$11 \to 16$	$11 \to 19$	$11 \to 16$	$11 \to 25$
012_H05	**012_H10**	**012_H15**	**012_H20**	**012_H25**	**012_H30**
$0 \to 3$	$0 \to 4$	$0 \to 6$	$0 \to \{13, 18, 24\}$	$0 \to 10$	$0 \to \{23, 26, 29, 34\}$
$1 \to 2$	$1 \to 3$	$1 \to 5$	$1 \to \{11, 21\}$	$1 \to \{12, 16\}$	$1 \to \{18, 38\}$
$2 \to 4$	$2 \to 6$	$2 \to \{11, 12, 14\}$	$2 \to \{14, 16, 19\}$	$2 \to \{13, 21, 26\}$	$2 \to \{21, 27, 35\}$
$3 \to 5$	$3 \to 5$	$3 \to 7$	$3 \to \{15, 17, 20, 26\}$	$3 \to \{14, 15\}$	$3 \to \{19, 28, 32\}$
$4 \to 6$	$\{11, 4\} \to 2$	$4 \to \{9, 13\}$	$\{4, 11\} \to 9$	$4 \to \{18, 20, 23, 24, 25\}$	$4 \to \{22, 31\}$
$5 \to 7$	$5 \to 7$	$5 \to 8$	$5 \to 10$	$5 \to 11$	$5 \to \{20, 25\}$
$6 \to 8$	$6 \to 8$	$6 \to 10$	$6 \to 12$	$6 \to \{17, 19\}$	$6 \to \{24, 30\}$
$7 \to 10$	$7 \to 9$	$7 \to 17$	$7 \to 23$	$7 \to 22$	$7 \to \varepsilon$
$8 \to 9$	$8 \to 10$	$8 \to 15$	$8 \to 22$	$8 \to \{29, 30\}$	$8 \to 39$
$9 \to 11$	$9 \to 11$	$9 \to 16$	$9 \to 25$	$9 \to \varepsilon$	$9 \to 33$
$10 \to 13$	$10 \to 12$	$10 \to 19$	$10 \to 27$	$10 \to 28$	$10 \to 36$
$11 \to 12$		$11 \to 18$		$11 \to 27$	$11 \to 37$

Table A.1: Ground truth of vertex-to-vertex correspondences between the models and their noisy symbols used in the experiments. The models and some examples of noisy symbols are presented in Table 5.13

Appendix B

Munkres assignment algorithm

This appendix presents how to find an optimal solution for an assignment problem based on the Munkres (or Hungarian) algorithm. To begin with, we give an example of the assignment problem and construct a cost matrix to present the problem solution. Next, we introduce the general steps of the Hungarian method and detail them in the example to obtain the optimal solution. Finally, we discuss the complexity of the Munkres algorithm. All the below information is referred from `https://brilliant.org/wiki/hungarian-matching/#citation-1`.

Assignment problem Suppose that you will have a wedding party for your son, and you need to rent a Wedding Catering and Wedding Food company. Each company can cover three principal services: logistics, food, and serving. Within your local residence, there are three possible options: A, B, and C. Nevertheless, one company can provide one service at a time, i.e., company A can not provide logistics and food at the same time. Therefore, the task is to match each company with a type of service for the wedding party. It is an assignment problem and can be presented as in Table B.1. From the table, the objective is to find an assignment between the companies and the services such that the total cost is minimized.

Company	Cost for logistic	Cost for food	Cost for serving
Company A	108	125	150
Company B	150	135	175
Company C	122	148	250

Table B.1: Quotations of three wedding catering and wedding food companies, prices are in euro (€).

Cost matrix From Table B.1, a cost matrix can be established as follows. This cost is called the *initial adjacency matrix*.

108	125	150
150	135	175
122	148	250

Munkres method The Munkres method operates on the following idea: if a number is added to or subtracted from all of the entries of any one row or column of a cost matrix, then an optimal assignment for the resulting cost matrix is also an optimal assignment for the original cost matrix. The main steps of the Munkres method are as follows.

1. Subtract the smallest entry in each row from all the other entries in the row. This will make the smallest entry in the row now equal to 0.
2. Subtract the smallest entry in each column from all the other entries in the column. This will make the smallest entry in the column now equal to 0.
3. Draw lines through the row and columns that have the 0 entries such that the fewest lines possible are drawn.
4. If there are n lines drawn, an optimal assignment of zeros is possible and the algorithm is finished. If the number of lines is less than n, then the optimal number of zeroes is not yet reached. Go to the next step.
5. Find the smallest entry not covered by any line. Subtract this entry from each row that is not crossed out, and then add it to each column that is crossed out. Then, go back to Step 3.

Calculation detail All the details of calculation steps are illustrated in Fig. B.1. So, the optimal assignment is Company A $\rightarrow$ serving, Company B $\rightarrow$ food, Company C $\rightarrow$ logistic, and the corresponding optimal cost is $150 + 135 + 122 = 407$ €.

Complexity of the Munkres algorithm Munkres algorithm is strongly polynomial. According to [Munkres, 1957], the original time complexity of Munkres algorithm is $\mathcal{O}(n^4)$, where n is the assignment variables. Later, Edmonds and Karp, and independently Tomizawa proved that it can have an $\mathcal{O}(n^3)$ running time [Edmonds and Karp, 1972; Tomizawa, 1971].

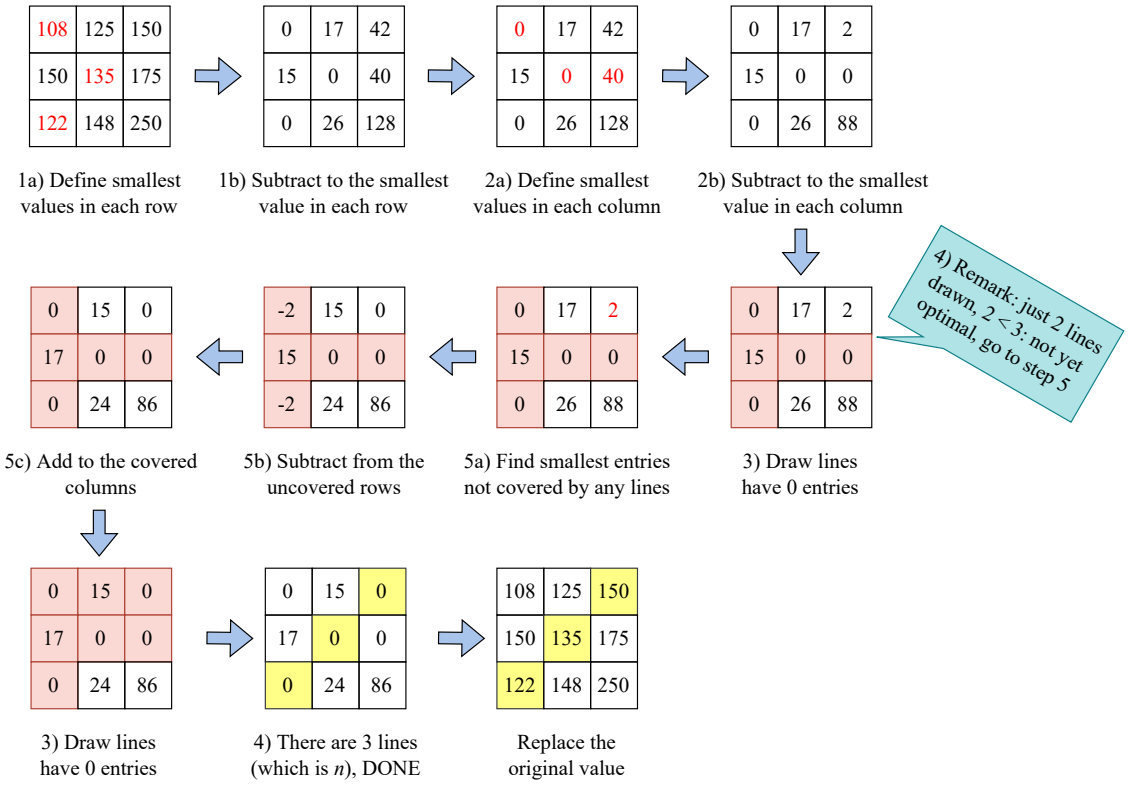


Figure B.1: Details of calculation steps of Munkres algorithm for finding the optimal solution for the assignment problem based on the cost matrix.

Bibliography

Z. Abu-Aisheh. *Approches anytime et distribueés pour l'appariment de graphes*. PhD thesis, University of Tours, Tours, 2016. URL `http://www.theses.fr/2016TOUR4024`.

Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, and P. Martineau. An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems. In *International Conference on Pattern Recognition Applications and Methods (ICPRAM)*, 2015.

Z. Abu-Aisheh, R. Raveaux, and J.-Y. Ramel. Anytime graph matching. *Pattern Recognition Letters*, 84:215–224, 2016.

T. Achary, S. Pillay, S. M. Pillai, M. Mqadi, E. Genders, and A. E. Ezugwu. A performance study of meta-heuristic approaches for quadratic assignment problem. *Concurrency and Computation: Practice and Experience*, page e6321, 2021.

J. Altschuler, J. Niles-Weed, and P. Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. *Advances in Neural Information Processing Systems (NIPS)*, 30, 2017.

R. Ambauen, S. Fischer, and H. Bunke. Graph edit distance with node splitting and merging, and its application to diatom identification. In *International Workshop on Graph-Based Representations in Pattern Recognition (GbRPR)*, pages 95–106, 2003.

R. Angles and C. Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1–39, 2008.

L. Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM Symposium on Theory of Computing (STOC)*, pages 684–697, 2016.

E. Bengoetxea, P. Larranaga, I. Bloch, and A. Perchant. Estimation of distribution algorithms: A new evolutionary computation approach for graph matching problems. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR)*, pages 454–469. Springer, 2001.

S. Berretti, A. Del Bimbo, and P. Pala. A graph edit distance based on node merging. In *International Conference on Image and Video Retrieval (ICIVR)*, pages 464–472, 2004.

S. Berretti, A. D. Bimbo, and P. Pala. Graph edit distance for active graph matching in content based retrieval applications. *The Open Artificial Intelligence Journal*, 1(1), 2007.

L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.

P. E. Black. Greedy algorithm. *Dictionary of Algorithms and Data Structures*, 2:62, 2005.

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

M. C. Boeres, C. C. Ribeiro, and I. Bloch. A randomized heuristic for scene recognition by graph matching. In *International Workshop on Experimental and Efficient Algorithms*, pages 100–113, 2004.

S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, and M. Vento. Graph edit distance as a quadratic assignment problem. *Pattern Recognition Letters*, 87:38–46, 2017.

J. F. Bravo Ferreira, Y. Khoo, and A. Singer. Semidefinite programming approach for the quadratic assignment problem with a sparse graph. *Computational Optimization and Applications*, 69(3):677–712, 2018.

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

E. T. Bullmore and D. S. Bassett. Brain graphs: graphical models of the human brain connectome. *Annual Review of Clinical Psychology*, 7:113–140, 2011.

H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, 1983.

T. Caelli and S. Kosinov. An eigenspace projection clustering method for inexact graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 26(4):515–519, 2004.

R. Cesar, E. Bengoetxea, and I. Bloch. Inexact graph matching using stochastic optimization techniques for facial feature recognition. In *Object recognition supported by user interaction for service robots*, volume 2, pages 465–468. IEEE, 2002.

R. M. Cesar Jr, E. Bengoetxea, I. Bloch, and P. Larrañaga. Inexact graph matching for model-based recognition: Evaluation and comparison of optimization algorithms. *Pattern Recognition*, 38(11):2099–2113, 2005.

P.-A. Champin and C. Solnon. Measuring the similarity of labeled graphs. In *International Conference on Case-Based Reasoning (ICCBR)*, pages 80–95, 2003.

M. Chertok and Y. Keller. Efficient high order matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(12):2205–2215, 2010.

M. Cho, K. Alahari, and J. Ponce. Learning graphs to match. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 25–32, 2013.

D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 18(03):265–298, 2004.

T. Cour, P. Srinivasan, and J. Shi. Balanced graph matching. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 19. MIT Press, 2007.

M. Darwiche. *When operations research meets structural pattern recognition : on the solution of error-tolerant graph matching problems.* PhD thesis, University of Tours, Tours, 2018. URL `http://www.theses.fr/2018TOUR4022`.

M. F. Demirci, A. Shokoufandeh, Y. Keselman, L. Bretzner, and S. Dickinson. Object recognition as many-to-many feature matching. *International Journal of Computer Vision (IJCV)*, 69(2):203–222, 2006.

M. F. Demirci, Y. Osmanlioglu, A. Shokoufandeh, and S. Dickinson. Efficient many-to-many feature matching under the l1 norm. *Computer Vision and Image Understanding*, 115(7):976–983, 2011.

A. Deruyver, Y. Hodé, E. Leammer, and J.-M. Jolion. Adaptive pyramid and semantic graph: Knowledge driven segmentation. In *International Workshop on Graph-Based Representations in Pattern Recognition (GbRPR)*, pages 213–222, 2005.

B. Ding, X.-y. Yu, and L. Liu. 3d cad model representation and retrieval based on hierarchical graph. *J. Softw.*, 9(10):2499–2506, 2014.

K. Dong, Z. Qin, and T. Wan. Many-to-one stable matching for prediction in social networks. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE)*, pages 345–356, 2020.

M. Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano*, 1992.

M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation (TEVC)*, 1(1):53–66, 1997.

M. Dorigo and T. Stützle. Ant colony optimization: overview and recent advances. *Handbook of metaheuristics*, pages 311–351, 2019.

M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial life*, 5(2):137–172, 1999.

F. Du, Q. Xuan, and T.-J. Wu. One-to-many node matching between complex networks. *Advances in Complex Systems (ACS)*, 13(06):725–739, 2010.

X. Du, J. Yan, and H. Zha. Joint link prediction and network alignment via cross-graph embedding. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2251–2257, 2019.

A. Dutta and A. Asaithambi. One-to-many bipartite matching based coalition formation for multi-robot task allocation. In *International Conference on Robotics and Automation (ICRA)*, pages 2181–2187, 2019.

J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

H. Ezzati, M. Amintoosi, and H. Tabasi. On the minimum of true matches in exact graph matching with simulated annealing. *Journal of Algorithms and Computation*, 53(1): 123–134, 2021.

L. Fan, Z. Wang, B. Cail, C. Tao, Z. Zhang, Y. Wang, S. Li, F. Huang, S. Fu, and F. Zhang. A survey on multiple object tracking algorithm. In *International Conference on Information and Automation (ICIA)*, pages 1855–1862, 2016.

Z. Fan, C. Mao, Y. Wu, and J. Xu. Spectral graph matching and regularized quadratic relaxations: Algorithm and theory. In *International Conference on Machine Learning (ICML)*, pages 2985–2995. PMLR, 2020.

M. Fey, J. E. Lenssen, C. Morris, J. Masci, and N. M. Kriege. Deep graph matching consensus. *arXiv preprint arXiv:2001.09621*, 2020.

A. Fischer, K. Riesen, and H. Bunke. Improved quadratic time approximation of graph edit distance by combining hausdorff matching and greedy assignment. *Pattern Recognition Letters*, 87:55–62, 2017.

E. Fix and J. L. Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247, 1989.

M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

H. Fröhlich, A. Košir, and B. Zajc. Optimization of fpga configurations using parallel genetic algorithm. *Information Sciences*, 133(3-4):195–219, 2001.

M. R. Gary and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness. *WH Freman and Co*, 1979.

M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

M. Gong, Y. Wu, Q. Cai, W. Ma, A. K. Qin, Z. Wang, and L. Jiao. Discrete particle swarm optimization for high-order graph matching. *Information Sciences*, 328:158–171, 2016.

Y.-J. Gong, E. Chen, X. Zhang, L. M. Ni, and J. Zhang. Antmapper: An ant colony-based map matching approach for trajectory-based applications. *IEEE Transactions on Intelligent Transportation Systems (T-ITS)*, 19(2):390–401, 2017.

S. Goss, S. Aron, J.-L. Deneubourg, and J. M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581, 1989.

A.-I. Grapa, L. Blanc-Féraud, E. Van Obberghen-Schilling, and X. Descombes. Optimal Transport vs Many-to-many assignment for Graph Matching. In *Colloque Francophone de Traitement du Signal et des Images (GRETSI)*, 2019.

M. Guo, E. Chou, D.-A. Huang, S. Song, S. Yeung, and L. Fei-Fei. Neural graph matching networks for fewshot 3d action recognition. In *Proceedings of the European conference on computer vision (ECCV)*, pages 653–669, 2018.

T. N. Ha, H. X. Huan, et al. An efficient ant colony optimization algorithm for multiple graph alignment. In *2013 International Conference on Computing, Management and Telecommunications (ComManTel)*, pages 386–391. IEEE, 2013.

F. Harary and R. Z. Norman. *Graph theory as a mathematical model in social science*. University of Michigan, Institute for Social Research Ann Arbor, 1953.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics (IEEE SMC)*, 4(2):100–107, 1968.

J. He, Z. Huang, N. Wang, and Z. Zhang. Learnable graph matching: Incorporating graph partitioning with deep feature learning for multiple object tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5299–5309, 2021.

J. F. Henriques, R. Caseiro, and J. Batista. Globally optimal solution to multi-object tracking with merged measurements. In *International Conference on Computer Vision (ICCV)*, pages 2470–2477, 2011.

N. Hiransakolwong, K. Vu, K. A. Hua, and S.-D. Lang. Many-to-many skeletal-graphs matching approach to shape recognition. In *Proc. International Conference: Sciences of Electronic, Technologies of Information and Telecommunications (IEEE SETIT)*. Citeseer, 2004.

B. Holzschuh, Z. Lähner, and D. Cremers. Simulated annealing for 3d shape correspondence. In *International Conference on 3D Vision (3DV)*, pages 252–260. IEEE, 2020.

J. Jans. On frobenius algebras. *Annals of Mathematics*, pages 392–407, 1959.

Y. Keselman, A. Shokoufandeh, M. F. Demirci, and S. Dickinson. Many-to-many graph matching via metric embedding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–I. IEEE, 2003.

K. Khurshid, C. Faure, and N. Vincent. A novel approach for word spotting using merge-split edit distance. In *International Conference on Computer Analysis of Images and Patterns (CAIP)*, pages 213–220, 2009.

J. Kim, H. Choi, H. Yun, and B.-R. Moon. Measuring source code similarity by finding similar subgraph with an incremental genetic algorithm. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 925–932, 2016.

D. Kimelman, M. Kimelman, D. Mandelin, and D. Yellin. Bayesian approaches to matching architectural diagrams. *Transactions on Software Engineering (TSE)*, 36(2):248–274, 2009.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

T. C. Koopmans and M. Beckmann. Assignment problems and the location of economic activities. *Econometrica: journal of the Econometric Society*, pages 53–76, 1957.

B. H. Korte, J. Vygen, B. Korte, and J. Vygen. *Combinatorial optimization*, volume 1. Springer, 2011.

S. Kpodjedo, P. Galinier, and G. Antoniol. Enhancing a tabu algorithm for approximate graph matching by using similarity measures. In *European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, pages 119–130. Springer, 2010.

S. Kpodjedo, F. Ricca, P. Galinier, G. Antoniol, and Y.-G. Gueheneuc. Madmatch: Many-to-many approximate diagram matching for design comparison. *IEEE Transactions on Software Engineering (TSE)*, 39(8):1090–1111, 2013.

S. Kpodjedo, P. Galinier, and G. Antoniol. Using local similarity measures to efficiently address approximate graph matching. *Discrete Applied Mathematics*, 164:161–177, 2014.

S. I. Ktena, S. Parisot, J. Passerat-Palmbach, and D. Rueckert. Comparison of brain networks with unknown correspondences. *arXiv preprint:1611.04783*, 2016.

G. Laporte, A. Asef-Vaziri, and C. Sriskandarajah. Some applications of the generalized travelling salesman problem. *Journal of the Operational Research Society (JORS)*, 47 (12):1461–1467, 1996.

C. O. Laura, S. Wesarg, and G. Sakas. Graph matching survey for medical imaging: On the way to deep learning. *Methods*, 2021.

E. L. Lawler. The quadratic assignment problem. *Management Science*, 9(4):586–599, 1963.

E. L. Lawler. The traveling salesman problem: a guided tour of combinatorial optimization. *Wiley-Interscience Series in Discrete Mathematics*, 1985.

P. Le Bodic, P. Héroux, S. Adam, and Y. Lecourtier. An integer linear program for substitution-tolerant subgraph isomorphism and its use for symbol spotting in technical drawings. *Pattern Recognition*, 45(12):4214–4224, 2012.

M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. In *Tenth IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1482–1489 Vol. 2, 2005.

M. Leordeanu, M. Hebert, and R. Sukthankar. An integer projected fixed point method for graph matching and map inference. *Advances in Neural Information Processing Systems (NIPS)*, 22, 2009.

Z.-Y. Liu and H. Qiao. Gnccp-graduated nonconvexityand concavity procedure. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 36(6):1258–1267, 2013.

J. Lladós, E. Martí, and J. J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 23(10):1137–1143, 2001.

E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research (EJOR)*, 176(2):657–690, 2007.

G. Lyu, S. Feng, T. Wang, C. Lang, and Y. Li. Gm-pll: graph matching based partial label learning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2019.

G. Lyu, S. Feng, and Y. Li. Partial multi-label learning via probabilistic graph matching mechanism. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 105–113, 2020.

J. Ma, X. Jiang, A. Fan, J. Jiang, and J. Yan. Image matching from handcrafted to deep features: A survey. *International Journal of Computer Vision (IJCV)*, 129(1):23–79, 2021.

H. P. Maretic, M. E. Gheche, M. Minder, G. Chierchia, and P. Frossard. Wasserstein-based graph alignment. *arXiv preprint arXiv:2003.06048*, 2020.

S. M. Meshkani and B. Farooq. Graph-based many-to-one dynamic ride-matching for shared mobility services in congested networks. *arXiv preprint arXiv:2101.08657*, 2021.

B. T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 20(5):493–504, 1998.

T. Mondal, N. Ragot, J.-Y. Ramel, and U. Pal. Flexible sequence matching technique: An effective learning-free approach for word spotting. *Pattern Recognition*, 60:596–612, 2016.

N. Monmarché, F. Guinand, and P. Siarry. *Artificial ants*. Wiley-ISTE Hoboken, 2010.

P. Morrison and J. J. Zou. Inexact graph matching using a hierarchy of matching processes. *Computational Visual Media*, 1(4):291–307, 2015.

D. Mostafa, C. Donatello, R. Romain, and T. Vincent. A local branching heuristic for solving a graph edit distance problem. *Computers & Operations Research*, 106:225–235, 2019. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2018.02.002.

J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics (JSTOR)*, 5(1):32–38, 1957.

M. Neuhaus and H. Bunke. *Bridging the gap between graph edit distance and kernel machines*, volume 68. World Scientific, 2007.

M. Neuhaus, K. Riesen, and H. Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In *Joint International Workshops on Statistical Techniques in Pattern Recognition and Structural and Syntactic Pattern Recognition (S+SSPR)*, pages 163–172, 2006.

A. Nowak, S. Villar, A. S. Bandeira, and J. Bruna. Revised note on learning quadratic assignment with graph neural networks. In *Data Science Workshop (DSW)*, pages 1–5. IEEE, 2018.

Y. Osmanlıoğlu and A. Shokoufandeh. Multilayer matching of metric structures using hierarchically well-separated trees. *Pattern Recognition Letters*, 87:63–70, 2017.

Y. Osmanlıoğlu, B. Tunç, D. Parker, M. A. Elliott, G. L. Baum, R. Ciric, T. D. Satterthwaite, R. E. Gur, R. C. Gur, and R. Verma. System-level matching of structural and functional connectomes in the human brain. *NeuroImage*, 199:93–104, 2019.

C. Park, T. J. Woehl, J. E. Evans, and N. D. Browning. Minimum cost multi-way data association for optimizing multitarget tracking of interacting objects. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 37(3):611–624, 2014.

A. A. Perera, C. Srinivas, A. Hoogs, G. Brooksby, and W. Hu. Multi-object tracking through simultaneous long occlusions and split-merge conditions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 666–673, 2006.

R. J. Qureshi, J.-Y. Ramel, H. Cardot, and P. Mukherji. Combination of symbolic and statistical features for symbols recognition. In *International Conference on Signal Processing, Communications and Networking (ICSPCN)*, pages 477–482, 2007.

L. Rastrigin. The convergence of the random search method in the extremal control of a many parameter system. *Automaton & Remote Control*, 24:1337–1342, 1963.

R. Raveaux. *Contributions and perspectives on combinatorial optimization and machine learning for graph matching and classification.* PhD thesis, Université de Tours, 2019.

P. Riba, A. Fischer, J. Lladós, and A. Fornés. Learning graph edit distance by graph neural networks. *Pattern Recognition*, 120:108132, 2021.

E. Rica, S. Álvarez, and F. Serratosa. On-line learning the graph edit distance costs. *Pattern Recognition Letters*, 146:55–62, 2021.

K. Riesen. Structural pattern recognition with graph edit distance. In *Advances in Computer Vision and Pattern Recognition*. Springer, 2015.

K. Riesen and H. Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition and Structural and Syntactic Pattern Recognition (S+SSPR)*, pages 287–297. Springer, 2008.

K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950–959, 2009.

K. Riesen and M. Ferrer. Predicting the correctness of node assignments in bipartite graph matching. *Pattern Recognition Letters*, 69:8–14, 2016.

K. Riesen, S. Fankhauser, and H. Bunke. Speeding up graph edit distance computation with a bipartite heuristic. In *MLG*, pages 21–24. Citeseer, 2007a.

K. Riesen, M. Neuhaus, and H. Bunke. Bipartite graph matching for computing the edit distance of graphs. In *International Workshop on Graph-Based Representations in Pattern Recognition (GbRPR)*, pages 1–12, 2007b.

K. Riesen, A. Fischer, and H. Bunke. Improving approximate graph edit distance using genetic algorithms. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition and Structural and Syntactic Pattern Recognition (S+SSPR)*, pages 63–72, 2014.

K. Riesen, A. Fischer, and H. Bunke. Improved graph edit distance approximation with simulated annealing. In *International Workshop on Graph-Based Representations in Pattern Recognition (GbRPR)*, pages 222–231. Springer, 2017.

M. Rolínek, P. Swoboda, D. Zietlow, A. Paulus, V. Musil, and G. Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. In *European Conference on Computer Vision (ECCV)*, pages 407–424. Springer, 2020.

O. Sammoud, C. Solnon, and K. Ghédira. Ant algorithm for the graph matching problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, pages 213–223, 2005.

O. Sammoud, S. Sorlin, C. Solnon, and K. Ghédira. A comparative study of ant colony optimization and reactive search for graph matching problems. In *European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, pages 234–246, 2006.

A. Sanfeliu and K.-S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics (IEEE SMC)*, 13(3):353–362, 1983.

P. Santacruz and F. Serratosa. Learning the graph edit costs based on a learning model applied to sub-optimal graph matching. *Neural Processing Letters*, 51(1):881–904, 2020.

P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4938–4947, 2020.

C. Schellewald and C. Schnörr. Probabilistic subgraph matching based on convex relaxation. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR)*, pages 171–186. Springer, 2005.

K. Schiff et al. An ant colony optimisation algorithm for the triple matching problem. *Czasopismo Techniczne*, 2:179–186, 2018.

G. L. Scott and H. C. Longuet-Higgins. An algorithm for associating the features of two images. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 244 (1309):21–26, 1991.

M. L. Sebu and H. Ciocarlie. Business process similarity metric supporting one-to-many relationship. In *Jubilee International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pages 429–435, 2015.

F. Serratosa. Fast computation of bipartite graph matching. *Pattern Recognition Letters*, 45:244–250, 2014.

F. Serratosa. Speeding up fast bipartite graph matching through a new cost matrix. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 29(02): 1550010, 2015.

F. Serratosa. Graph edit distance: restrictions to be a metric. *Pattern Recognition*, 90: 250–256, 2019.

F. Serratosa and X. Cortés. Graph edit distance: Moving from global to local structure to solve the graph-matching problem. *Pattern Recognition Letters*, 65:204–210, 2015.

L. G. Shapiro and R. M. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, PAMI-3(5):504–519, 1981.

W. Shen, J. Shen, and H. Zheng. Fingerprint recognition system by use of graph matching. In *Object Detection, Classification, and Tracking Technologies*, volume 4554, pages 141–146. International Society for Optics and Photonics, 2001.

D. Shin, T. Kim, J. Choi, and J. Kim. Author name disambiguation using a graph model with node splitting and merging based on bibliographic information. *Scientometrics*, 100 (1):15–50, 2014.

A. Shokoufandeh, Y. Keselman, M. F. Demirci, D. Macrini, and S. Dickinson. Many-to-many feature matching in object recognition: a review of three approaches. *IET Computer Vision*, 6(6):500–513, 2012.

S. Sorlin and C. Solnon. Reactive tabu search for measuring graph similarity. In *International Workshop on Graph-Based Representations in Pattern Recognition (GbRPR)*, pages 172–182, 2005.

S. Sorlin, C. Solnon, and J.-M. Jolion. A generic graph distance measure based on multivalent matchings. In *Applied Graph Theory in Computer Vision and Pattern Recognition*, pages 151–181. Springer, 2007.

M. Stauffer, T. Tschachtli, A. Fischer, and K. Riesen. A survey on applications of bipartite graph edit distance. In *International Workshop on Graph-Based Representations in Pattern Recognition (GbRPR)*, pages 242–252, 2017.

T. Stützle and H. H. Hoos. Max–min ant system. *Future generation computer systems*, 16 (8):889–914, 2000.

N. Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1(2):173–194, 1971.

P. H. Torr. Solving markov random fields using semi definite programming. In *International Workshop on Artificial Intelligence and Statistics*, pages 292–299. PMLR, 2003.

W.-H. Tsai and K.-S. Fu. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Transactions on Systems, Man, and Cybernetics (IEEE SMC)*, 9(12):757–768, 1979.

S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 10(5):695–703, 1988.

N. Vesselinova, R. Steinert, D. F. Perez-Ramirez, and M. Boman. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 8: 120388–120416, 2020.

M. Vlastelica, A. Paulus, V. Musil, G. Martius, and M. Rolínek. Differentiation of blackbox combinatorial solvers. *arXiv preprint arXiv:1912.02175*, 2019.

R. Wang, J. Yan, and X. Yang. Learning combinatorial embedding networks for deep graph matching. In *International Conference on Computer Vision (ICCV)*, pages 3056–3065, 2019.

Y. Wang and N. Ishii. A genetic algorithm and its parallelization for graph matching with similarity measures. *Artificial Life and Robotics*, 2(2):68–73, 1998.

R. J. Wilson. *Introduction to graph theory*. Pearson Education India, 1979.

Y. Wu, M. Gong, W. Ma, and S. Wang. High-order graph matching based on ant colony optimization. *Neurocomputing*, 328:97–104, 2019.

Z. Xu, P. Chen, R. Raveaux, X. Yang, and H. Liu. Deep graph matching meets mixed-integer linear programming: Relax at your own risk? *arXiv preprint arXiv:2108.00394*, 2021.

A. Yadav et al. Discrete artificial electric field algorithm for high-order graph matching. *Applied Soft Computing*, page 106260, 2020.

J. Yan, X.-C. Yin, W. Lin, C. Deng, H. Zha, and X. Yang. A short survey of recent advances in graph matching. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, pages 167–174, 2016.

J. Yan, S. Yang, and E. R. Hancock. Learning for graph matching and related combinatorial optimization problems. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4988–4996, 2020.

X. Yang. Metaheuristic Optimization. *Scholarpedia*, 6(8):11472, 2011.

A. Zanfir and C. Sminchisescu. Deep learning of graph matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2684–2693, 2018.

M. Zaslavskiy, F. Bach, and J.-P. Vert. A path following algorithm for the graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 31(12):2227–2242, 2008.

M. Zaslavskiy, F. Bach, and J.-P. Vert. Many-to-many graph matching: a continuous relaxation approach. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pages 515–530. Springer, 2010.

Z. Zhang, H. Che, P. Shi, Y. Sun, and J. Gu. Multi-labeled graph matching–an algorithm model for schema matching. In *Annual Asian Computing Science Conference*, pages 90–103, 2005.

Z. Zhang, P. Shi, H. Che, Y. Sun, and J. Gu. Formulation schema matching problem for combinatorial optimization problem. *Int. J. Interoperability Bus. Inf. Syst.*, 1:33–60, 2006.

Z. Zhang, Y. Xiang, L. Wu, B. Xue, and A. Nehorai. Kergm: Kernelized graph matching. *Advances in Neural Information Processing Systems*, 32, 2019.

F. Zhou and F. De la Torre. Factorized graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 38(9):1774–1789, 2015.

List of Publications

International Conferences/Workshops

- Kieu-Diem Ho, Jean-Yves Ramel and Nicolas Monmarché. Multivalent Graph Matching for Symbol Recognition. In International Conference on Document Analysis and Recognition (ICDAR) (pp. 488-503). Springer, Cham. 2021
- Kieu-Diem Ho, Jean-Yves Ramel and Nicolas Monmarché. Multivalent graph matching problem solved by max-min ant system. In Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR) (pp. 227-237). Springer, Cham. 2021.

National Conferences

- Kieu-Diem Ho, Jean-Yves Ramel and Nicolas Monmarché. Comparaison de contenus d'images par appariement multivoque de graphes. In ORASIS, 2021.

Kieu Diem HO

Comparaison de Données Structurelles à l'Aide d'Algorithmes Evolutionnaires

Résumé

Les graphes sont des structures de données capables de décrire des entités complexes. Leurs propriétés leur procurent un intérêt particulier pour de nombreuses applications comme la bio-informatique, la vision par ordinateur ou l'analyse des réseaux sociaux. Les sommets du graphe correspondent aux sous-parties des objets et les arcs représentent les relations entre les sous-parties. Le problème de comparaison d'objets devient alors un problème de mise en correspondance de graphes. Dans ce domaine, la plupart des travaux traitent de l'appariement univalent entre graphes et peu d'études abordent le problème de l'appariement multivalent. Pour palier à cette lacune, dans cette thèse, nous fournissons d'abord une définition claire et précise de l'appariement multivalent de graphes basée sur une formulation étendue de la distance d'édition entre graphes. Ensuite, nous proposons d'utiliser la méta-heuristique d'optimisation par colonies de fourmis (ACO) pour résoudre ce problème combinatoire. Une stratégie de recherche locale est appliquée pour accélérer le temps de calcul de l'algorithme ACO. La distance d'édition étendue proposée est évaluée sur un ensemble de données d'images de symboles, démontrant de bonnes performances pour les tâches de classification et pour la mise en correspondance de sous-parties d'objets dans des contextes bruités.

Mots-clés : comparaison de graphes, appariement de graphes, distance d'édition sur graphes, méta-heuristiques, algorithme évolutionnaire, optimisation par colonies de fourmis, classification.

Abstract

Graphs are data structures capable of describing complex entities. Their properties make them particularly interesting for many applications such as bioinformatics, computer vision or social network analysis. The vertices of the graph correspond to the subparts of the objects and the arcs represent the relations between the subparts. The object comparison problem then becomes a graph matching problem. In this field, most of the works deal with univalent matching between graphs and few studies address the problem of multivalent matching. To overcome this gap, in this thesis, we first provide a clear and precise definition of multivalent graph matching based on an extended formulation of the edit distance between graphs. Then, we propose to use ant colony optimization (ACO) meta-heuristics to solve this combinatorial problem. A local search strategy is applied to speed up the computation time of the ACO algorithm. The proposed extended edition distance is evaluated on a symbol image dataset, demonstrating good performance for classification tasks and for matching object subparts in noisy contexts.

Keywords: multivalent graph matching, graph edit distance, meta-heuristics, evolutionary algorithm, ant colony optimization, classification