Contributions and perspectives on combinatorial optimization and machine learning for graph matching and classification

Romain Raveaux

HABILITATION À DIRIGER DES RECHERCHES

En INFORMATIQUE

Titre en français :
Contributions et perspectives sur l'optimisation combinatoire et l'apprentissage pour l'appariement et la classification de graphes
Titre en anglais :
Contributions and perspectives on combinatorial optimization and machine learning for graph matching and classification

Année universitaire : 2018 / 2019

présentée et soutenue publiquement par :

Romain RAVEAUX

le 26 Juin 2019

RAPPORTEURS :

M. Luc	BRUN	Professeur des universités	ENSI Caen
M. Frederico	DELLA CROCE	Professeur	Ecole Polytechnique de Turin, Italie
M. Kaspar	RIESEN	Professeur	University of Applied Sciences FHNW, Olten, Suisse
M. Francesc	SERRATOSA	Professeur	Université de Tarragone, Espagne

JURY :

M. Luc	BRUN	Professeur des universités	ENSI Caen
M. Frederico	DELLA CROCE	Professeur	Ecole Polytechnique de Turin, Italie
M. Josep	LLADOS	Professeur	Université Autonome de Barcelona, Espagne
M. Jean-Yves	RAMEL	Professeur des universités	Université de Tours
M. Franscesc	SERRATOSA	Professeur	Université de Tarragone, Espagne
Mme Christine	SOLNON	Professeur des universités	INSA Lyon
Mme Nicole	VINCENT	Professeur des universités	Université Paris Descartes

Remerciements

Je tiens à exprimer tout d'abord mes remerciements aux membres du jury, qui ont accepté d'évaluer mon travail d'HDR. Merci à Mme Christine Solnon, Professeur des universités, à l'INSA Lyon, d'avoir accepté de présider le jury de cette HDR. Merci à messieurs les Professeurs Kaspar Riesen de l'université des sciences appliquées FHNW de Suisse, Franscesc Serratosa de l'université de Tarragone en Espagne, Frederico Della Croce de l'école polytechnique de Turin en Italie et Luc Brun de l'ENSI Caen d'avoir accepté d'être les rapporteurs de ce manuscrit. Leurs remarques et suggestions lors de la lecture de mon rapport m'ont permis d'apporter des améliorations à la qualité de ce dernier. Merci également aux professeurs Nicole Vincent de l'université de Paris et Josep Llados de l'université de Barcelone, pour avoir accepté d'examiner mon mémoire et de faire partie de mon jury de thèse. Merci à Jean-Yves Ramel pour avoir accepté d'être le référent de cette HDR, son regard et ses suggestions ont été une aide précieuse pour la construction de ce manuscrit. Je tiens également à le remercier pour la confiance et la sympathie qu'il m'a témoignée.

Merci aux doctorants que j'ai eu la chance de co-encadrer, Zeina Abu-Aisheh, Mostafa Darwiche et Maxime Martineau. Ces collaborations ont mené à un enrichissement mutuel.

Je tiens à remercier aussi l'équipe RFAI du LIFAT sans laquelle tout cela n'aurait été possible. Une autre équipe m'a grandement aidé dans l'élaboration de mon HDR, l'équipe ROOT du LIFAT avec laquelle j'ai aimé collaborer. Merci au LIFAT de m'avoir permis de travailler dans de bonnes conditions. Merci à Polytech'Tours de l'université de Tours de m'avoir soutenu dans ma démarche.

Finalement, j'adresse un grand merci à toute ma famille qui a toujours été présente lorsque j'en ai eu besoin, en particulier à ma compagne Laetitia et mon fils.

Résumé

Mots clés: Appariement de graphes, classification de graphes, apprentissage, optimisation combinatoire, reconnaissance des formes, vision par ordinateur.

Problématique de recherche

La problématique de recherche abordée dans ce manuscrit est la suivante:
Comment mettre en correspondance, mesurer des similarités et classifier des objets lorsque les objets sont représentés par des graphes?

En d'autres termes, comment comparer deux graphes? Comment sont reliés graphes et classification? Comment extraire l'information contenue dans des graphes? Les graphes sont des structures de données informatiques très souples qui permettent une description très riche et très fine d'une gamme très large d'objets, allant des molécules chimiques aux images, en passant par des réseaux sociaux. Le revers de la médaille est que comparer des graphes peut s'avérer une tâche d'une grande complexité calculatoire. La complexité et l'optimisation combinatoire sont liées à une discipline entière appelée la Recherche opérationnelle. En recherche opérationnelle, les problèmes d'optimisation doivent être formalisés et bien structurés. Un algorithme d'optimisation exploite ces informations structurées pour les résoudre. Au contraire, l'apprentissage automatique est un domaine de recherche traitant de la conception d'algorithmes permettant de résoudre des problèmes au moyen d'approches statistiques. Les algorithmes d'apprentissage automatique extraient des informations à partir d'exemples pour faire des prédictions, ce qui constitue une différence majeure par rapport aux algorithmes de recherche opérationnelle. Le cœur des algorithmes d'apprentissage réside dans leur capacité à apprendre et à généraliser à partir d'informations "non structurées" ou "non formalisées". Mes travaux concernent l'**apprentissage** et l'**optimisation combinatoire** pour la mise en correspondance et la classification de graphes dans des problèmes de **reconnaissance des formes** et de **vision par ordinateur**.

Abstract

Keywords: Graph matching, graph classification, structural learning, combinatorial optimisation, pattern recognition, computer vision.

Research problems
How to match, measure similarities and classify objects when objects are represented by graphs?

Graphs are very flexible computer data structures that allow a very rich and very detailed description of a very wide range of objects, ranging from chemical molecules to images, via social networks. However, the reverse of the medal is often an increase of computational complexity. Complexity and combinatorial optimization are related to an entire discipline called operations research. In operations research, optimization problems need to be formalized and well structured. An optimization algorithm exploits this structured information to solve it. At the opposite, machine learning is a research field dealing with the design of algorithms for solving problems by means of statistical approaches. Machine learning algorithms learn from examples to make predictions, which is a major difference with operations research algorithms. The core of machine learning algorithms is their ability to learn and generalize from "unstructured" or "not formalized" information. The tricptic machine learning, combinatorial optimization and pattern recognition is still very open to new contributions and interesting discoveries remain to be realized. My research activities are focused around **combinatorial optimization** and **machine learning** to match and classify graphs for **pattern recognition** and **computer vision problems**.

Résumé long en français

Les problèmes de classification et d'appariement de graphes peuvent être traités sous deux angles différents que sont l'apprentissage et l'optimisation combinatoire. Ces deux visions radicalement opposées peuvent être réunies pour servir un objectif commun et résoudre un même problème. L'hybridation des deux domaines, optimisation combinatoire et apprentissage, conduit à de nouvelles méthodes de résolution ainsi qu'à de nouvelles perspectives. Le manuscrit est découpé en trois parties, la première traitant de l'optimisation combinatoire et la deuxième de l'apprentissage. Chacune de ces parties est elle-même découpée en deux sous-parties traitant de l'appariement et de la classification de graphes. Une dernière partie porte sur les perspectives évoquant notamment les interactions entre optimisation combinatoire et apprentissage.

Optimisation discrète pour l'appariement et la classification de graphes

Dans cette section sont discutées les problématiques liées au calcul de dissimilarité de graphes ainsi qu'à la classification de graphe basée sur ces dissimilarités.

Comparaison de graphes

De nombreuses applications, comme par exemple la recherche ou la classification d'informations, nécessitent de mesurer la distance ou la similarité entre deux graphes, i.e., apparier –mettre en correspondance– les sommets des graphes afin d'identifier leurs points communs et leurs différences. Il existe différents types d'appariements de graphes donnant chacun lieu à une définition différente de la distance entre deux graphes. Les appariements exacts (isomorphisme de graphes ou de sous-graphe) permettent de montrer que deux graphes sont identiques ou qu'un graphe est inclus dans un autre graphe. Cependant, dans de nombreuses applications, supposer l'existence d'un tel appariement est une hypothèse trop forte. Par conséquent, des appariements de graphes tolérants aux erreurs tels que la recherche du plus grand sous-graphe commun à deux graphes ou la distance d'édition de graphes ont été proposés. L'appariement recherché est alors un "meilleur" appariement, i.e., un appariement devant préserver le plus grand nombre de sommets et d'arcs des graphes sans pour autant nécessairement tous les préserver. L'idée de la distance d'édition de graphes est de définir la similarité de deux graphes par le nombre minimal d'opérations élémentaires d'édition nécessaires pour transformer un graphe en un autre. La distance d'édition de graphes est calculée par un ensemble standard d'opérations d'édition, i.e. les insertions de nœuds ou d'arcs, les suppressions de nœuds ou d'arcs et les substitutions de nœuds ou d'arcs. En outre, une fonction de coûts est associée à chacune de ces opérations et l'objectif est de trouver l'ensemble des opérations qui minimise la somme des coûts d'édition. Nous nous sommes intéressés au problème du calcul de la distance d'édition entre graphes qui fournit à la fois un appariement et une mesure de dissimilarité entre deux graphes. Un premier objectif a été de définir de nouveaux modèles mathématiques pour représenter le problème de la distance d'édition entre graphes. Lors d'une collaboration avec le LITIS et dans la thèse de Mostafa Darwiche, trois modèles fondés sur la programmation linéaire en nombres entiers (PLNE) ont été élaborés. La PLNE est une manière de décrire un problème par une fonction de coût, des contraintes linéaires et par des variables entières. Ce formalisme permet de bénéficier de méthodes de résolution efficaces facilement exploitables grâce à des solveurs. Un solveur est un logiciel informatique capable de résoudre des équations mathématiques ou des problèmes de logique. La distance d'édition entre graphes est un problème d'optimisation NP-Difficile. Son temps de résolution croit de manière exponentielle en fonction du nombre de nœuds des deux graphes. Par conséquent, deux défis apparaissent pour

ce type de problème. Tout d'abord, l'élaboration de méthodes exactes permettant d'obtenir la solution optimale du problème de manière rapide. La résolution exacte n'est pas toujours possible en pratique du fait de l'explosion combinatoire engendrée par la complexité du problème. De ce constat né le deuxième défi, la conception de méthodes heuristiques capables de fournir rapidement une solution sous optimale de qualité. Nous avons proposé deux méthodes de résolutions exactes calculant la solution optimale du problème d'optimisation. Dans la thèse de Zeina Abu-aisheh une recherche arborescente de type séparation et évaluation a été proposée. Une évaluation de toutes les solutions possibles est exécutée sans les énumérer explicitement. Les solutions partielles sont éliminées à l'aide des bornes inférieures et supérieures. Dans la thèse de Mostafa Darwiche l'utilisation d'un solveur mathématique a permis de résoudre les trois formulations basées sur la PLNE. Le couple PLNE et solveur mathématique a permis d'obtenir les meilleurs résultats. Etant donné que la résolution exacte n'est pas toujours possible en pratique, nous nous sommes intéressés à la résolution heuristique. Dans ma thèse, j'ai exploré la possibilité de simplifier le problème initiale pour le transformer en un problème d'affectation de sous graphes dont la résolution s'opère en temps polynomiale et non plus en temps exponentiel comme le problème de départ. Bien sûr, ce gain de temps ne se fait pas sans conséquence sur la qualité de la solution obtenue. There is no free lunch [1]. Dans la thèse de Zeina Abu-aisheh et dans la collaboration avec le LITIS, des heuristiques sont obtenues simplement en limitant en temps l'exécution des méthodes exactes. Ce faisant, il est aisé de répondre aux contraintes de temps de certaines applications mais aucune information sur la qualité de la solution retournée n'est prise en compte pour stopper la méthode. Cet inconvénient est levé dans la thèse de Mostafa Darwiche, deux recherches locales, au sens d'un opérateur de voisinage dans l'espace des solutions, s'appuyant sur la PLNE et un solveur mathématique ont été proposées. Ces méthodes explorent l'espace des solutions localement autour d'un voisinage et s'arrêtent si aucune solution améliorante n'est trouvée. Pour finir, les solveurs mathématiques, comme IBM Cplex par exemple, sont en évolutions constantes et deviennent d'année en année de plus en plus efficaces. Cet élément laisse à penser que les approches développées par le LIFAT vont encore gagner en efficacité dans l'avenir. Une prise de recul sur les méthodes de résolution du problème de la distance d'édition entre graphes a permis de rapprocher les notions de méthode exacte et heuristique en proposant les méthodes dites anytime. Ce type de méthode est capable de délivrer une première solution réalisable très rapidement pour ensuite l'améliorer progressivement jusqu'à converger vers une solution optimale. A chaque fois qu'une solution améliorante est trouvée, elle est mise à disposition pour l'application finale qui utilise la méthode anytime comme un service de production de solutions. Cette manière d'appréhender le problème rend la méthode anytime très flexible et applicable lorsque l'on ne connait pas à l'avance les contraintes de temps de l'application finale. Pour finir, une base méthodologique solide composée de bases de graphes et de métriques a été proposée pour l'évaluation de performance des méthodes de résolution du problème de la distance d'édition entre graphes. De cette démarche est né un concours sur ce problème dans le cadre de la conférence internationale en reconnaissance des formes ICPR 2016 en collaboration avec des collègues du laboratoire GREYC de Caen.

Classification de graphes basée distance

Cette partie aborde la problématique de classification supervisée de graphes. Dans de nombreuses applications, il est en effet nécessaire d'affecter une classe (catégorie) à un graphe inconnu. Cette étape de classification s'appuie sur un ensemble de graphes dont la classe est connue. Cet ensemble de graphes est appelé base d'apprentissage. La littérature propose principalement deux types d'approches pour résoudre un problème de classification supervisée de graphes: les approches à base de noyaux et des approches de type K Plus Proches Voisins (KPPV). Cette dernière est la plus fréquemment adoptée pour sa simplicité de mise en œuvre et ses bonnes performances.

[1] Phrase de l'économiste Milton Friedman

Elle ne nécessite pas d'apprentissage mais simplement le stockage des données d'apprentissage (TrS=base d'apprentissage). Une nouvelle donnée (ou requête) de classe inconnue est comparée à toutes les données de la base d'apprentissage. Pour la requête, l'algorithme choisit la classe majoritaire parmi ses K plus proches voisins dans la base d'apprentissage au sens d'une distance choisie.

Cette méthode dans le cadre des graphes souffre toutefois d'un défaut majeur qu'est la complexité calculatoire. Ce défaut apparait pour deux raisons: 1) la méthode des KPPV nécessite de calculer pour chaque graphe requête g toutes les distances $d(g, g') \quad \forall g' \in TrS$. 2) Dans le contexte des graphes, calculer $d(g, g')$ est un problème NP-Difficile. Trois angles méthodologiques différents corrigent ces défauts: 1) L'utilisation d'une heuristique rapide du problème d'édition de graphes pour calculer $d(g, g')$ dans un contexte de KPPV. Cette solution a été usitée dans la thèse de Zeina Abu-aisheh ainsi que dans ma thèse. 2) Il est aussi possible de "réduire" la base d'apprentissage de graphes, en sélectionnant ou en générant des représentants à partir de la base initiale. Dans ma thèse et dans une collaboration avec le LITIS, je m'étais intéressé au calcul de graphe médian (modélisant une classe) et de graphe prototype (discriminant). Ces aspects de sélection ou de génération de prototypes relèvent de la section apprentissage qui sera développée après. 3) Enfin, le dernier angle d'attaque consiste à modéliser le problème du calcul des KPPV comme un problème d'optimisation discret et à le résoudre de manière heuristique. Cette méthodologie orignale a été validée dans un travail avec Zeina Abu-aisheh. La modélisation du problème du calcul des KPPV a été réalisée en généralisant à plusieurs graphes le problème de la distance d'édition. Cette approche rend alors possible la réalisation d'un algorithme de type séparation et évaluation qui élimine des comparaisons de graphes non prometteuses grâce aux bornes supérieures et inférieures calculées sur le problème des KPPV. Il en résulte une exploration optimisée de l'arbre des comparaisons.

Apprentissage dans l'espace des graphes pour l'appariement et la classification de graphes

Dans cette section est discutée l'élaboration et l'utilisation de méthode d'apprentissage pour résoudre des problèmes d'appariement et de classification de graphes.

Appariement de graphes

Dans la thèse de Maxime Martineau, le problème de la distance d'édition entre graphes a été paramétré. Les paramètres w viennent pondérer les fonctions de coûts entre deux nœuds ou deux arcs ainsi le problème de mise en correspondance "nœud à nœud " ou "arc à arc" est dépendant de paramètres continus (réels). L'algorithme d'apprentissage a pour objectif de trouver les valeurs des paramètres qui minimisent un critère défini sur les données de la base d'apprentissage (risque empirique). Un exemple de critère peut être de minimiser la somme des erreurs au carré entre les appariements obtenus par une heuristique (paramétrée) et une méthode exacte. Le but étant d'avoir une heuristique aussi précise qu'une méthode exacte. Ce problème s'apparente à un problème de régression structurée. Un outil efficace pour résoudre un problème de régression est la modélisation par réseau de neurones et l'entrainement de ce dernier par la méthode de la descente de gradient. Dans contexte, l'apprentissage consiste à trouver les valeurs des paramètres w. La distance d'édition paramétrée peut être vue comme une couche particulière appelée couche combinatoire qui est intégrable dans un réseau de neurones. Cette couche combinatoire nécessite un hyper-paramètre: un graphe modèle/prototype qui doit être fixe pendant la phase d'apprentissage. L'inconvénient de cette méthode est qu'elle ne permet pas de généraliser les paramètres appris à plusieurs graphes prototypes. Les paramètres w sont donc associés à un graphe prototype en particulier.

Classification de graphes

L'approche précédente a été adaptée pour classifier des graphes. Le critère à minimiser est une erreur de classification sur la base d'apprentissage. Dans le cadre d'un problème à deux classes, la sortie de la couche combinatoire vient alimenter une fonction d'activation afin de prédire la classe du graphe à classifier. Les entrées de la couche combinatoire restent similaires à l'approche précédente: un graphe prototype fixe durant la phase d'apprentissage. La question du calcul du graphe prototype a été évoquée sur la partie concernant le problème des KPPV. Dans ma thèse et en collaboration avec le LITIS, nous avions défini quatre types de graphe prototype: 1) les graphes d'ensemble appartenant à la base d'apprentissage, 2) les prototypes généralisés pouvant ne pas appartenir à la base d'apprentissage, les graphes généralisés sont des graphes synthétiques issus d'un processus de génération. Chacune de ces deux grandes familles se scindent en deux parties: 3) les graphes médians sont construits au regard d'une seule et unique classe du problème de classification sans prendre en considération les autres classes. Par opposition, 4) les graphes discriminants sont construits en tenant compte de toute la base d'apprentissage dans un objectif de minimiser une erreur de classification. Dans l'approche mentionnée précédemment, le prototype de graphe est choisi *a priori* en calculant un graphe médian d'ensemble. Dans la thèse de Maxime Martineau, l'architecture précédente de type perceptron a été étendue au concept de réseau de neurones convolutifs (CNN). Cette stratégie a permis de s'abstraire des prototypes de graphes calculés *a priori* pour le bénéfice de graphes discriminants déterminés pendant la phase d'apprentissage. L'ensemble de ces travaux constitue des bases solides pour étendre les réseaux de neurones profonds (deep learning) à des données représentées sous forme de graphe.

Perspectives pour l'appariement et la classification de graphes

Dans les problèmes d'appariement et de classification de graphes, l'apprentissage peut être intégré à différents niveaux allant de l'apprentissage des graphes (structures et attributs) en passant par les mesures de similarité entre nœuds et arcs jusqu'à l'exploration de l'arbre de recherche composé de tous les appariements possibles entre graphes. Des perspectives sont évoquées comme les couches combinatoires dans les réseaux profonds, le transport optimal pour l'appariement de graphes et l'apprentissage par renforcement. Finalement, des applications sont décrites comme l'adaptation de domaine non supervisée et le tracking d'objets dans des vidéos.

Contents

List of figures xiii

List of tables xvi

I CV et Activités scientifiques 1

II Contributions and perspectives on combinatorial optimization and machine learning for graph matching and classification 3

1 General introduction 5

2 Notations, problems and applications 9

2.1 Graphs: types and definitions . . . 9

2.2 Graph matching problems . . . 11

2.2.1 Exact isomorphism . . . 11

2.2.2 Error-tolerant problems . . . 12

2.2.3 Graph Edit Distance . . . 13

2.3 Structural pattern recognition problems . . . 15

2.3.1 Graph-based search . . . 15

2.3.2 Graph classification . . . 16

2.3.3 Graph analytic problems . . . 17

2.4 Applications and representations . . . 17

2.4.1 Applications . . . 17

2.4.2 Graph-based representations . . . 19

2.4.3 Why pattern recognition based on graphs ? . . . 20

2.5 Surveys and organization . . . 20

3 Combinatorial optimization for graph matching and graph classification 23

3.1 Graph matching . . . 24

3.1.1 State of the art . . . 25

3.1.2 Open problems . . . 46

3.1.3 Contribution . . . 47

3.1.4 Summary . . . 61
3.2 Graph classification . . . 62
3.2.1 State of the art . . . 62
3.2.2 Open problems . . . 67
3.2.3 Contribution . . . 67
3.2.4 Summary . . . 78

4 Structural machine learning for graph matching and graph classification 79
4.1 Graph matching . . . 80
4.1.1 State of the art of learning graph matching . . . 80
4.1.2 Open problems . . . 92
4.1.3 Contributions . . . 93
4.1.4 Summary . . . 100
4.2 Graph classification . . . 101
4.2.1 State of the art from a machine learning viewpoint . . . 101
4.2.2 Open problems . . . 106
4.2.3 Contributions . . . 107
4.2.4 Summary . . . 118

5 Conclusion and perspectives 119
5.1 Conclusion . . . 119
5.1.1 Discrete optimization for graph matching and graph classification . . . 119
5.1.2 Graph matching and graph classification in graph space . . . 121
5.1.3 Interplay between machine learning and combinatorial optimization . . . 122
5.2 Short term perspectives . . . 123
5.2.1 Learning-free graph matching problems . . . 123
5.2.2 Learning-based graph matching . . . 125
5.2.3 Learning graph matching for classification . . . 125
5.3 Long term perspectives . . . 128
5.3.1 Bringing semantic to computer vision task thanks to graphs . . . 128
5.3.2 On the relation between graph matching and Optimal Transport . . . 128
5.3.3 Graph matching for domain adaptation . . . 129
5.3.4 The rise of the edge classification . . . 129
5.3.5 Benchmarking . . . 129
5.3.6 Deep reinforcement learning . . . 131
5.3.7 Graph matching for multi-object tracking in videos and document analysis 131
5.3.8 Scalability . . . 132
5.4 Synthesis . . . 132

Bibliography 132

III Appendix **153**

A More graph matching problems **155**
A.1 Graph isomorphism . . . 155
A.2 Monomorphism . . . 155
A.3 Substitution-Tolerant Subgraph Isomorphism . . . 156
A.4 Multivalent Matching . . . 156

B Machine learning theory **159**
B.1 The basics of losses . . . 160
B.1.1 Cross-entropy loss . . . 160
B.1.2 Mean square error loss . . . 162
B.1.3 Regularization of loss functions . . . 163
B.2 Probably Approximately Correct (PAC) . . . 164
B.3 Structured prediction . . . 164

C Graph neural networks **167**
C.1 History . . . 167
C.2 The basics of artificial neural networks . . . 167
C.3 The basics of graph neural networks . . . 168
C.3.1 Definitions . . . 168
C.3.2 Intuition . . . 169
C.3.3 A simple example for $f(.,.)$: . . . 169
C.3.4 More complex message aggregations . . . 170
C.4 Applications and losses . . . 172
C.4.1 Unsupervised . . . 172
C.4.2 Supervised . . . 172
C.4.3 Semi-Supervised . . . 173
C.4.4 Applications . . . 174
C.5 Advanced GNN . . . 176
C.5.1 Gated GNN . . . 176
C.5.2 Graph pooling . . . 178
C.5.3 Differentiation and training . . . 178
C.5.4 A word on scalability and time complexity? . . . 179
C.5.5 Spatial vs spectral convolution on graphs . . . 179
C.6 Summary on GNN . . . 181

D Kernels and kernel machines for graphs **183**
D.1 Kernel theory . . . 183
D.1.1 Kernel and dissimilarity . . . 184
D.2 Kernel machines . . . 184
D.2.1 Distance to the decision line . . . 184
D.2.2 Finding the margin . . . 184

D.2.3 Maximizing the margin . . . 185
D.2.4 Rescaling the parameters . . . 185
D.2.5 Canonical formulation . . . 185
D.2.6 Lagrangian formulation . . . 186
D.2.7 Dual formulation and kernel formulation . . . 186
D.2.8 Solving SVM . . . 186
D.2.9 Classification . . . 187
D.3 Graph mathing-based Kernels . . . 187
D.3.1 Trivial GED kernels . . . 187
D.3.2 Kernels based on GED embedding [Riesen and Bunke, 2010b] . . . 188
D.3.3 Kernel from Maximum-Similarity Edit Path . . . 188
D.3.4 Convolution kernels and local matching kernels [Neuhaus and Bunke., 2007] 188
D.3.5 Random Walk Edit Kernel [Neuhaus and Bunke., 2007] . . . 189
D.3.6 Diffusion kernel . . . 190

E Publications **191**
E.1 Liste des publications . . . 191
E.1.1 Thèse . . . 191
E.1.2 Chapitre de livres . . . 191
E.1.3 Revues Internationales . . . 191
E.1.4 Revues Nationales . . . 193
E.1.5 Conférences Internationales de rang A (CORE 2018) . . . 193
E.1.6 Conférences Internationales de rang B (CORE 2018) . . . 193
E.1.7 Autres conférences Internationales . . . 194

List of Figures

1.1 A functional block diagram of a pattern recognition process. The whole procedure maps the pattern domain to the label domain. . . . 6

3.1 An example of graph matching modelled as a QAP. . . . 26
3.2 An example of graph matching modelled as a CRF. . . . 27
3.3 A relation between the Graph Edit Distance (GED), Quadratic Assignment (QAP), Maximum-a posteriori (MAP) inference and error-tolerant graph matching (ETSGM and ECGM) problems. . . . 28
3.4 Example of graph matching and related matrices [Zhou and la Torre, 2016]. (a) Two synthetic graphs. (b) The correspondence matrix X. (c) The first graph's incidence matrix G_1. (d) The second graph's incidence matrix G_2. (e) The node affinity matrix K_p. (f) The edge affinity matrix K_q. (g) The global affinity matrix K. . . . 34
3.5 State of the art on graph matching.#papers is the number of papers according to the criteria: model type and heuristic or exact methods. A paper is cited for each category to give an example. . . . 36
3.6 State of the art on graph matching.#papers is the number of papers according to the type methods. A paper is cited for each category to give an example. . . . 38
3.7 A comparison of the sugbraph matching and error-correcting graph matching problems. The solutions 1 are the best in the two cases. . . . 39
3.8 A comparison of the sugbraph matching and error-correcting graph matching problems when deletion and insertion costs are identical for any vertex or edge and higher than any substitution cost. . . . 40
3.9 From [Zhang et al., 2016a]. The x-axis is the rotation angle gap between frames. The "Acc" axis represents the similarities between matchings. The normalized Obj axis represents the normalized objective function values. . . . 44
3.10 From [Zhang et al., 2016a]. Matching results on VOC images. Typical matching result are shown on the left. Yellow lines indicate correct matches, blue lines indicates incorrect matches, and green lines indicate matches between outliers. The results of DD are not shown due to the prohibitive execution time . . . 45
3.11 Graph data repository for error correcting graph matching . . . 52
3.12 Average speed-deviation scores on MUTA Dataset. . . . 53
3.13 Average speed-deviation scores on GREC Dataset. . . . 54
3.14 Local branching flow. a) depicts the left and right branching. b) shows the neighborhoods in the solution space . . . 55
3.15 Characteristics of anytime algorithms . . . 56

3.16 A random pair of graphs taken from MUTA-70 database illustrating the improvement of found solutions with the delight of time . . . 57
3.17 MUTA deviation: left (up to 40 ms), right (up to 400 ms). . . . 58
3.18 A comparison of the sugbraph matching and error-correcting graph matching problems when the similarity function $s(i \to k) = -\{c(i \to k) - c(i \to \epsilon) - c(\epsilon \to k)\}$. 60
3.19 Four graphs and the distance between each graph. . . . 65
3.20 Speeding up the kNN problem : State of the art of kNN methods operating in dissimilarity space. . . . 67
3.21 Histograms showing τ_b distribution for each heuristic for MUTA-30 (a) and PAH (b) 70
3.22 Multi graph edit distance in a single picture. . . . 71
3.23 *one-Tree-kMGED* when $k = 1$. Given a query graph G_q and graphs in the training set, the problems GED(G_q,G_1), GED(G_q,G_2) and GED(G_q,G_3) are considered as sub-trees of the global tree (T_q). The sub-tree of GED(G_q,G_j) is pruned thanks to *UB* that is found via GED(G_q,G_1). . . . 73
3.24 Scatter plots of the optimal edit distances (x-axis) and the suboptimal edit distances (y-axis). Orange dots are optimal distances. . . . 76

4.1 An overview of the learning graph matching problem. Three sub-problems arise as learning features, learning dissimilarity and learning to match. . . . 81
4.2 A general framework for supervised learning of graph matching. . . . 81
4.3 Local parametrization of graph matching. . . . 84
4.4 Siamese architecture: a structural definition. . . . 86
4.5 Siamese architecture for graph matching. . . . 87
4.6 Computational pipeline of the fully trainable graph matching model. . . . 89
4.7 A literature review of machine learning techniques for graph matching. . . . 90
4.8 Illustration of the parametrized score function computation. . . . 94
4.9 Parametrized matching function where G^1 has 7 graph components (4 nodes and 3 edges). . . . 95
4.10 Computation graph of our learning problem from the inputs to the loss. . . . 97
4.11 Principle of the gradient descent. . . . 98
4.12 Matching evolution from iteration 0 (top) to 200 (bottom). . . . 99
4.13 Training: Accuracy in function of the number of epochs. . . . 99
4.14 Machine learning techniques for graph classification and a focus on bridging the gap between graph matching and embedding techniques. . . . 101
4.15 Taxonomy of graph prototypes. . . . 103
4.16 Bridging the gap between end-to-end graph embedding and graph space techniques by learning graph matching and prototypes. . . . 107
4.17 Overview of the perceptron and a modified perceptron for graph classification. I in this figure is an input vector. . . . 108
4.18 Computation graph of our graph-based classifier from the inputs to the loss. . . . 109
4.19 Letter-HIGH : Impact of the learning rate on the convergence. . . . 112
4.20 Top: Original images. Bottom: Results of a filter graph convolves on regular grids of images (8-connexity). The neighbourhood is a one hop neighbourhood. . . . 114

4.21 Two graphs are convolved. G_I is the input graph and G_F is the filter graph. Attributes of G_F are parameters W. Convolution is based on matching G_F at different locations of the input graph. 114
4.22 Graph matching based convolution layer. 115
4.23 MNIST digits classiffication (Regular grid, Superpixels) (Taken from CVPR 2017 tutorial http://geometricdeeplearning.com/). 117

5.1 The combinatorial optimization (CO) solves a machine learning problem. 122
5.2 The combinatorial optimization (CO) algorithm repeatedly queries the same ML model to make decisions. The ML model takes as input the current state of the combinatorial algorithm. 123
5.3 Machine learning acts alone to provide a solution to the problem. 123
5.4 Extension of the graph matching problem to a higher order 124
5.5 A proposal of end-to-end learning graph matching scheme based on a combinatorial layer. Red parts are new components compared to [Nowak et al., 2017] 126
5.6 The ML method extracts information from the problem. This information (features) is fed to a combinatorial optimization (CO) algorithm. 126
5.7 A branch and bound algorithm guided by a learned predictor. 127
5.8 A proposal of end-to-end learning graph classification in the graph space. The left part of the image (the GNN) is taken from [Kipf and Welling, 2016] 127
5.9 Illustration of the domain adaptation. (left) dataset for training, i.e. source domain, and testing, i.e. target domain. Note that a classifier estimated on the training examples clearly does not fit the target data. (middle) a matching Y is estimated and used to transfer the training samples onto the target domain.(right) the transferred labeled samples are used for estimating a classifier in the target domain. 130
5.10 A mind map about the perspectives 133

B.1 Selection of the best parameters and meta parameters. 160

C.1 A time line about graph neural networks. 168
C.2 A general overview of a GNN. 170
C.3 Statistics on citation networks (From a T. Kipf's talk http://deeploria.gforge.inria.fr/thomasTalk.pdf). 174
C.4 Structured matrix completion (From CVPR 2017 tutorial http://geometricdeeplearning.com/). 174
C.5 MNIST digits classiffication (Regular grid, Superpixels) (From CVPR 2017 tutorial http://geometricdeeplearning.com/). 175
C.6 A general overview of a Gated GNN. 177

D.1 Orthogonal projection of the point A on the line. 185

List of Tables

1 Table of notations xvii

3.1 Graph matching literature 37
3.2 Dataset description for graph matching assessments. 43
3.3 Results on PAH dataset of error-correcting graph matching methods with a time limit of 300s. 45
3.4 Results on CMU House dataset of error-correcting graph matching methods with a time limit of 10s. 45
3.5 Cost functions for PAH and CMU House data sets. 46
3.6 Overview about the subsets included in the repository 52
3.7 Methods included in the graph edit distance contest of ICPR 2016. 53
3.8 A summary of kNN methods. 66
3.9 Percentage of the number of times, the H0 hypothesis ($\tau_b = 0$) was rejected for each heuristic on PAH instances 69
3.10 Percentage of the number of times, the H0 hypothesis ($\tau_b = 0$) was rejected for each heuristic on MUTA-30 instances 69
3.11 Data sets selected from the ICPR'2016 contest. 75
3.12 Learning-free cost functions. 75
3.13 Classification rate obtained by exact and heuristic methods. 75
3.14 The characteristics of the datasets included in the experiments. 77
3.15 The cost functions and meta parameters of the datasets. 77
3.16 Classification results on five datasets where *time* refers to the average time in milliseconds needed by each dissimilarity computation whereas *Acc* refers to the classification accuracy. The best results are marked in bold style. Note that k was fixed to 1. 78

4.1 Synthesis of the literature on learning graph matching. The double horizontal lines separate between shallow and deep methods. 91
4.2 Taken from [Leordeanu et al., 2012]: Comparison of matching rates for 2 graph matching algorithms before and after unsupervised learning on Cars and Motorbikes from Pascal07 database, with all outliers from the right image allowed and no outliers in the left image. 92
4.3 Taken from [Cho et al., 2013]: Performance on synthetic point sets. A learning approach and a learning-free method (shown in each row) are evaluated with the state of the art graph matching algorithms (in columns) 92

4.4 Test: Mean matching accuracy on unseen graphs. 98
4.5 Categorization of high level methods for comparing graphs. 102
4.6 Summary of graph data set characteristics. 111
4.7 Best parameters according to learning strategies R-1-NN and C-1-NN taken from **(alias?)**: [Riesen and Bunke, 2010b] and from **(alias?)**: [Moreno-García et al., 2016] 111
4.8 Taxonomy of the compared methods 112
4.9 Classification results. The best classification rates are marked in blue while the best processing times are in red. "Best" means an improvement over other methods statistically significant according to a z-test with a confidence level of 70% 113
4.10 Recognition rates on MNIST 2class 117

C.1 Properties of the Bioinformatics and Social network datasets used in graph/node classification experiments. 176
C.2 Test set classification accuracies (%). 176

Table of notations

$G = (V, E, \mu, \zeta)$	An attributed graph
u_i or i	Refers to vertices in a graph
i and j	Refers to vertices in the graph G_1
k and l	Refers to vertices in the graph G_2
$e = (i, j)$ or e_{ij} or ij	Refers to an edge in a graph
$\mu(i)$	Returns the attributes assigned to the vertex
$\zeta(e)$	Returns the attributes assigned to the edge
$\|.\|$	Cardinality of a set
n_1	Number nodes of the graph G_1
n_2	Number nodes of the graph G_2
N	Number nodes of a graph G
$\mathcal{G}$	Graph space
M	Size of the training set
m	Number of classes
TrS	Training set
TeS	Test set
$\mathcal{D}$	A set of graphs
t	A target value : a ground truth value, a class label, ...
$\hat{t}$	A prediction.
$c()$	A matching cost function.
Y	A node-to-node matching matrix
y	A node-to-node matching vector
p	A partial solution or a feasible solution
$\|\|.\|\|_2$	Norm L2
$Pr(X)$	Probability of X

Table 1: Table of notations

Part I

CV et Activités scientifiques

Part II

Contributions and perspectives on combinatorial optimization and machine learning for graph matching and classification

Chapter 1

General introduction

According to C. Bishop [Bishop, 2006], **pattern recognition** (PR) has its origins in engineering, whereas machine learning grew out of computer science. The field of PR is the term given to the science of automating the decisions on data such as classifying them into different categories. Humans are faced with a great diversity of pattern recognition problems in their everyday life. Examples of pattern recognition tasks, which are in the majority of cases intuitively solved, include the recognition of a written or a spoken word, the face of a friend, an object on the table, a traffic sign on the road, and many others. These simple examples illustrate the essence of PR. In the world, there exist classes of patterns which are recognized by humans according to certain knowledge learned before. The terminology pattern refers to any observation in the real world (e.g., an image, an object, a symbol, or a word, to name just a few). Roughly, pattern recognition is the assignment of a label to a given input value. In statistics, discriminant analysis was introduced for this purpose by Ronald Fisher in 1936. However, its mathematical foundations even go back to the 18th century (with Bayes, Laplace, Euler, etc.). Pattern classification can be a nontrivial problem due to the wide variability of patterns. For instance, image classification could be tackled using handcrafted rules or heuristics for distinguishing the visual patterns based on shape, color or texture. In practice such an approach would lead to a proliferation of rules and exceptions to the rules and so on, and could give poor results. An alternative solution can be obtained by adopting a **machine learning** (ML) approach. For S. Shwartz et al [Shalev-Shwartz and Ben-David, 2014], the term machine learning refers to the automated detection of meaningful patterns in data. For example, a large set of images along with their category is called a training set. This training set can be used to tune the parameters of an adaptive model during a so called learning phase. An algorithm that is capable to adapt the parameters of a model from the data is named a learning algorithm. During the learning phase, sub-patterns are detected, selected and combined to better take decisions. The traditional PR pipeline is depicted in Figure 1.1. Feature extraction and decision steps are coupled and can be trained together. PR is one approach to Artificial Intelligence [1] which underpins developments in cognate fields such as computer vision [2], image processing, text and document analysis for instance. Since in recent years, our world has become increasingly "digitized" and the amount of data available is dramatically increasing. In twenty years, our world has seen the explosion of Internet, the emergence of social networks, the

[1] 1956: Dartmouth workshop, first occurence of the term AI. "We propose a study of artificial intelligence [...]. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it" John McCarthy, Marvin L. Minsky, Nathaniel Rochester, and Claude E. Shannon

[2] It is commonly accepted that the father of Computer Vision is Larry Roberts (L. Roberts, "Machine perception of 3D solids", Chapter 9 in J. T. Tipp), who in his Ph.D. thesis (cir. 1960) at MIT discussed the possibilities of extracting 3D geometrical information from 2D perspective views of blocks. Computer vision has later emerged as a student summer project at MIT proposed by Papert, Seymour A https://dspace.mit.edu/handle/1721.1/6125

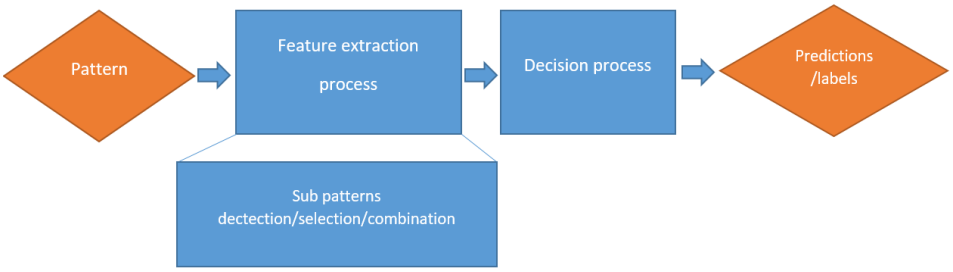


Figure 1.1: A functional block diagram of a pattern recognition process. The whole procedure maps the pattern domain to the label domain.

cloud, the Digital Humanities, the 4.0 Industry. Today, even fridges send spam[3]. Some examples:

- Worldwide, there are over 2.23 billion monthly active Facebook users (2018)
- The number of smartphone users is forecast to grow from 2.1 billion in 2016 to around 2.5 billion in 2019
- 100 terabytes of data uploaded daily to Facebook servers (2017).
- 600 million of photos, 200 millions of voice messages and 100 millions of video messages are posted on WhatsApp ... each day (2014)
- In 2016, IDC and SAP [4] predicted that 60 percent of global manufacturers would use analytics data recorded from connected devices to analyze processes and identify optimization possibilities. (2017)

At the same time, the explosion of computing power imagined by Moore [5] is one source of transformations in machine learning. Driven by this huge amount of data, the computational power and recent algorithmic progress, PR techniques has entered in a new era during the last 10 years. But with great power comes great responsibility as mentioned by Cedric Villani and al[6], it is important to structure and regulate the use of such powerful techniques. More than ever, Humanity (the quality of being humane) must be placed at the heart of the technological progress.

The question how to represent patterns in a formal way such that they can automatically be processed by machine is a key issue in pattern recognition and related fields. In general, there are two major ways to tackle this crucial step, viz., the statistical and the structural approach. In the statistical approach, feature vectors are employed for representing the underlying patterns. Vectors do not provide a direct possibility to describe relationships that might exist among different parts of a pattern. The use of structural data structures (i.e. strings, trees, or graphs) in PR, has created the branch of Structural Pattern Recognition (SPR). Among the data structures used, the one with the largest number of contributions, conferences and special issues of international journals is the representation based on graphs. Their growing popularity in the fields of PR can be explained by the ability of graphs to represent complex shapes across their ability to model simpler component interactions. However, we have to pay a price when using this type of enriched and interesting representations: computational complexity. Complexity and combinatorial optimization are related to an entire discipline called **Operations Research** [Korte and Vygen, 2007]. Operations

[3] https://www.tomshardware.fr/articles/internet-objet-frigo-spam,1-46695.html
[4] http://digitalistmag.wpengine.netdna-cdn.com/files/2016/03/IDC_IoT_white_paper_Mar2016.pdf
[5] Moore's law is the observation that the number of transistors in a dense integrated circuit doubles about every two years
[6] https://www.aiforhumanity.fr/pdfs/9782111457089_Rapport_Villani_accessible.pdf

research is a research field dealing with modelling and solving combinatorial optimization problems. In operations research, optimization problems need to be formalized and well structured. Mathematical programming, classically used when dealing with such problems, is a perfect example: a problem is completely modelled by variables, constraints and objectives to optimize. An optimization algorithm exploits this structured information to solve it. At the opposite, machine learning is a research field dealing with the design of algorithms for solving problems by means of statistical approaches. ML algorithms learn from examples to make predictions, which is a major difference with operations research algorithms. The core of ML algorithms is their ability to learn and generalize from "unstructured" or "not formalized" information. The tricptic machine learning, combinatorial optimization and pattern recognition make SPR still very open to new contributions and interesting discoveries remain to be realized. Thus, more than twenty years ago, a community has formed around the problematic of SPR. This community was built at the national level around groups such as AFRIF (French Association for Pattern Recognition and Interpretation) and at the international level around technical committees TC15 [7] (Graph-based representation for pattern recognition), TC2 (structural and syntactical pattern recognition) of International Association for Pattern Recognition (IAPR).

Graph-based representations have been used with considerable success for solving many problems in Document Image Analysis. For instance, my PhD thesis entitled "Graph Mining and Graph Classification: Application to cadastral map analysis." was established in this context. The rise of graph-based pattern recognition methods has been greatly supported by the document image analysis community. At the national level thank to Written Communication Research Group (GRCE) and at the international level through TC10 Graphic Recognition (GREC) and TC11 Reading Systems. Therefore, the dissemination is conveyed by a high number of scientific journals that are concerned with this research area (e.g., Pattern Analysis and Applications, International Journal of Document Analysis and Recognition (both Springer), Pattern Recognition and Pattern Recognition Letters (both Elsevier), and the IEEE Transactions on Pattern Analysis and Machine Intelligence, to name just a few examples. Conferences and workshop are also devoted to this topic Graph-based Representation (GbR), Structural/Syntactic Pattern Recognition (SSPR) workshops and a dedicated track on the International Conference on Pattern Recognition (ICPR). Finally, it is interesting to name some recent competitions on this field: Graph Distance Contest (2016), Competition on Subgraph Spotting in Graph Representations of Comic Book Images (2018) and Contest on Graph Matching Algorithms for Pattern Search in Biological Databases (2014) hosted by the ICPR. Subgraph Isomorphism challenge launched by the MIT (http://graphchallenge.mit.edu). Regionally speaking, problems related to graph computation are well represented at the ICVL Federation (Informatique Centre Val de Loire) and the research axis called Graph algorithmics and exponential complexity (18 members from 3 teams).

My work concerns the structural pattern recognition, proposing contributions related respectively to supervised graph classification and graph matching. Especially, a focus is given on similarity and dissimilarity computation on graphs. My work is applied to image analysis problems for object recognition and detection as well as molecule classification (Chemoinformatics). My scientific course is at the confluence of two research areas: combinatorial optimization and machine learning. By confronting and combining theses two visions, new proposals and a better understanding of structural pattern recognition problems have arisen. Since my first research activities started with my doctoral studies, I was led to model problems, to formalize and design algorithms. I have followed this thread to tackle problems in SPR. Of course, various colleagues contributed inestimably to this work. It would not be possible without the hard work of "my" PhD students (Zeina Abu-Aisheh, Mostafa Darwishe and Maxime Martineau) and my colleagues (Donatello Conte, Jean-Yves Ramel, Vincent T'kindt, Gilles Venturini, Antoine Tabbone, Alireza Alaei, Jean-Marc Ogier, Jean-Christophe Burie, Pierre Héroux and Sébastien Adam).

[7] https://iapr-tc15.greyc.fr/

The manuscript is organized as follows: Section 2 is dedicated to definitions and notations necessary to introduce the problems of graph matching and classification. In Section 3, state of the art, deadlocks and contributions on the two aforementioned problems are presented through the view point of combinatorial optimization. On the contrary, in Section 4, graph matching and classification problems are addressed in the light of machine learning. Short and long term perspectives are presented in Section 5.

Chapter 2

Notations, problems and applications

The objective of this section is to present basic notions about the graph theory. Notations used in the rest of the document are introduced, problems and applications are identified. This section is organized as follows. Formal definitions of graphs are presented in Section 2.1. Mathematical definitions of the graph matching problems are given in Section 2.2. A categorization of the structural pattern recognition problems are proposed in Section 2.3. Then, applications of SPR are put forward in Section 2.4. Finally, the main surveys on SPR problems are mentioned and the organization of the manuscript is explained in Section 2.5.

2.1 Graphs: types and definitions

Graphs are an efficient data structure and the most general formalism for object representation in structural Pattern Recognition (PR). They are basically composed of a finite or infinite set of vertices V, that represents parts of objects, connected by a set of edges $E \subseteq V \times V$, that represents the relations between these two parts of objects, where each edge connects two vertices in the graph. Formally saying, $e = (u_i, u_j)$, or e_{ij}, where both u_i and u_j are vertices that belong to the set V.

Definition 1. *Graph*
$G = (V, E)$
V is a set of vertices
E is a set of edges such that $E \subseteq V \times V$

A graph G is said to be *undirected* when each edge e_{ij} of the set E has no direction. This kind of graphs represents a symmetric relation. Mathematically saying: $(u_i, u_j) \in E = (u_j, u_i) \in E$. In contrast to the *directed* graphs which respect the direction that is assigned to each edge e_{ij}. Thus, for the *directed* graphs $(u_i, u_j) \neq (u_j, u_i)$. Non-attributed graphs are only based on their neighborhood structures defined by edges. Thus, no attributes can be found on neither the edges nor the vertices of graphs. Whereas in attributed, or labelled, graphs (AG), significant attributes can be found on edges, vertices or both of them which efficiently describe objects (in terms of shape, color, coordinate, size, etc.) and their relations.

In AGs, four extra components have been added (L_V,L_E,μ,ζ) to take into account vertex and edge attributes.

Mathematically speaking, AG is considered as a set of 6 tuples (V,E,L_V,L_E,μ,ζ) such that:

Definition 2. *Attributed Graph*
$G = (V,E,L_V,L_E,\mu,\zeta)$

V is a set of vertices
E is a set of edges such as $E \subseteq V \times V$
L_V is a set of vertex attributes
L_E is a set of edge attributes
$\mu : V \to L_V$. μ is a vertex labeling function which associates the label l_{u_i} to a vertex u_i
$\zeta : E \to L_E$. ζ is an edge labeling function which associates the label $l_{e_{ij}}$ to an edge e_{ij}

In the literature and to make the notation simpler, L_V and L_E are omitted which leads to $G = (V,E,\mu,\zeta)$.

Definition 2 allows to handle arbitrarily structured graphs with unconstrained labeling functions. For example, attributes of both vertices and edges can be part of the set of integers $L = \{1, 2, 3, \cdots\}$, the vector space $L = \mathbb{R}^n$ and/or a finite set of symbolic attributes $L = \{x, y, z, \cdots\}$.

In PR, a combination of both symbolic and numeric attributes on vertices and edges is required in order to describe the properties of vertices and their relations. For notational convenience, directed attributed relational graphs are simply referred to as graphs in the rest of the manuscript.

Graph size Let $G = (V,E,\mu,\zeta)$ be a graph. In this manuscript, the size of G is the number of nodes in V. The size of G is denoted by $N = |V|$. Let $G_1 = (V_1,E_1,\mu_1,\zeta_1)$ and $G_2 = (V_2,E_2,\mu_2,\zeta_2)$ be two graphs. The size of G_1 is denoted by $n_1 = |V_1|$. The size of G_2 is denoted by n_2. When two graphs have the same size then we use the notation $N = n_1 = n_2$.

Walk, circuit, path and cycle

1. **Walk:** A walk is a sequence of vertices and edges of a graph. Vertices can be repeated and edges can be repeated.
2. **Trail** A Walk in which no edge is repeated. Vertices can be repeated and edges cannot be repeated.
3. **Circuit:** is a closed trail. Vertices can be repeated and edges are not repeated.
4. **Path:** It is a trail in which neither vertices nor edges are repeated.
5. **Cycle:** Traversing a graph such that no vertex and no edge are repeated but the starting and ending vertex must be same i.e. starting and ending vertices can be repeated. Vertices are not repeated and edges are not repeated.

Adjacency matrix The adjacency matrix representation of a graph is $|V| \times |V|$ matrix A where $A_{ij} = 1$ if $e_{ij} \in E$ and 0 otherwise. For undirected graphs the matrix A is symmetric and $A^T =$ A. Walks of length n can be computed by looking at the n-th power of A (A^n).

Degree matrix The out-degree of a node u_i, $\omega^+(u_i)$ is equal to $\omega^+(u_i) = \sum_{(u_i,u_j)\in E} A_{ij}$. The in-degree of a node u_i, $\omega^-(u_i)$ is equal to $\omega^-(u_i) = \sum_{(u_j,u_i)\in E} A_{ji}$. Note that in an undirected graph, $\omega^+(u_i) = \omega^-(u_i) \forall u_i \in V$ and is denoted $\omega(u_i)$. The out-degree matrix Ω^+ is a diagonal matrix with $\Omega^+_{ii} = \omega^+(u_i)$, and similarly for Ω^-. When the graph is undirected, one has $\Omega^+ = \Omega^- = \Omega$. $\tilde{A} = \Omega^{-1}A$ is a stochastic matrix such that each row sums to one. $\tilde{A}^n_{ij}$ gives the probability of walks of length n from node i to node j.

Laplacian For undirected graphs, several Laplacian formulation exist. The combinatorial Laplacian is the matrix $\nabla = \Omega - A$. The normalized Laplacian is $\tilde{\nabla} = \Omega^{-1/2}\nabla\Omega^{-1/2}$. The random walk Laplacian $\nabla^{rw} = \Omega^{-1} - \nabla$. More details on Laplacian are given in [Zhou and Schölkopf, 2005].

Once we have defined these basic notions of graph theory, we can define the graph matching problems in the next section.

2.2 Graph matching problems

Graph matching (GM) is the process of finding a correspondence between the vertices and the edges of two graphs that satisfies some (more or less stringent) constraints ensuring that similar substructures in one graph are mapped to similar substructures in the other. Matching problems are divided into two broad categories: the first category contains exact GM problems that require a strict correspondence among the two objects being matched or at least among their subparts. The second category defines error-tolerant GM problems, where a matching can occur even if the two graphs being compared are structurally different to some extent. GM, whether exact or error-tolerant, is applied on patterns that are transformed into graphs. This approach is called structural in the sense of using the structure of the patterns to compare them.

2.2.1 Exact isomorphism

2.2.1.1 Induced subgraph isomorphism

Induced Subgraph Isomorphism is the problem of finding a subgraph (G_1) in a larger graph (G_2). More formally, when comparing two graphs $G_1 = (V_1,E_1,\mu_1,\zeta_1)$ and $G_2 = (V_2,E_2,\mu_2,\zeta_2)$, we are looking for an function $f : V_1 \to V_2$ which maps each vertex $u_i \in V_1$ onto a vertex $u_j \in V_2$ such that certain conditions are fulfilled :

Problem 1. *Induced SubGraph Isomorphism (ISGI)*
An injective function $f : V_1 \to V_2$ is a subgraph isomorphism from G_1 to G_2 if:

1. $\forall u_i \in V_1,\ \mu_1(v) = \mu_2(f(u_i))$
2. $\forall u_i, u_j \in V_1, (u_i, u_j) \in E_1 \Leftrightarrow (f(u_i), f(u_j)) \in E_2$
3. $\forall (u_i, u_j) \in E_1,\ \zeta_1((u_i, u_j)) = \zeta_2((f(u_i), f(u_j)))$

The $\mathcal{NP}$-completeness proof of subgraph isomorphism can be found in [Garey and Johnson, 1990].

2.2.1.2 Maximum Common Subgraph (MCS)

Maximum Common Subgraph is the problem of mapping a subgraph of the source graph to an isomorphic subgraph of the target graph. Usually, the goal is to find the largest subgraph for which such a mapping exists.

Problem 2. *Maximum Common Subgraph (MCS)*
Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. A graph $G_s = (V_s, E_s)$ is said to be a common subgraph of G_1 and G_2 if there exists subgraph isomorphism from G_s to G_1 and from G_s to G_2. The largest common subgraph is called the maximum common subgraph, or MCS, of G_1 and G_2.

2.2.2 Error-tolerant problems

Error-tolerant problems is a family of graph matching problems that contains two important members error-tolerant subgraph matching and error-correcting graph matching.

2.2.2.1 From exact to error-tolerant: problem transformation

The stringent constraints imposed by exact GM are, in some circumstances, too rigid for the comparison of two graphs. So the matching process must be tolerant: it must accommodate the differences by relaxing, to some extent, the constraints that define the matching type.

Error-tolerant matching is generally needed when no significant identical part of the structure together with the corresponding vertex and edge attributes in graphs G_1 and G_2 can be found. Instead, matching G_1 and G_2 is associated to a penalty cost. For example, this case occurs when vertex and edge attributes are numerical values (scalar or vectorial). The penalty cost for the mapping can then be defined as the sum of the distances between label values. A first solution to tackle such problems relies on a discretization or a classification procedure to transform the numerical values into nominal/symbolic attributes. The main drawback of such approaches is their sensitivity to frontier effects of the discretization or misclassification. A subsequent exact GM algorithm would then be unsuccessful. A second solution consists in using exact GM algorithms and customizing the compatibility function for pairing vertices and edges. The main drawback of such approaches is the need to define thresholds for these compatibilities. A last way consists in using an error-tolerant GM procedure that overcomes this drawback by integrating the numerical values during the mapping search. In this case, the matching problem turns from a decision one to an optimization one.

2.2.2.2 Error-Tolerant Subgraph Matching

Error-Tolerant Subgraph Matching [Messmer and Bunke, 1998] takes into account the difference in topology as well as attributes. Thus, it requires that each vertex/edge of graph G_1 is mapped to a distinct vertex/edge of graph G_2 or to a dummy vertex/edge. This dummy elements can absorb structural modifications between the two graphs. ε_{V_1} is a set of dummy vertices of G_2 such that $|\varepsilon_{V_1}| = |V_1|$ and $\mathcal{V}_2 = V_2 \cup \varepsilon_{V_1}$. Similarly, ε_{E_1} is a set of dummy edges of G_2 and $\mathcal{E}_2 = E_2 \cup \varepsilon_{E_1}$.

Problem 3. *Error-Tolerant SubGraph Matching (ETSGM)*
An injective function $f : V_1 \to \mathcal{V}_2$ is an error-tolerant subgraph isomorphism from $G_1 = (V_1,E_1,\mu_1,\zeta_1)$ to $G_2 = (\mathcal{V}_2,\mathcal{E}_2,\mu_2,\zeta_2)$
Function f must satisfy certain conditions:

1. $\forall u_i \in V_1, f(u_i) \in \mathcal{V}_2$
2. $\forall u_i, u_j \in V_1, (u_i, u_j) \in E_1 \Rightarrow (f(u_i), f(u_j)) \in \mathcal{E}_2$
3. $\forall u_i \in V_1, \mu_1(u_i) \approx \mu_2(f(u_i))$ *and* $\forall (u_i, u_j) \in E_1, \zeta_1((u_i, u_j)) \approx \zeta_2((f(u_i), f(u_j)))$

2.2.2.3 Error-Correcting Graph Matching

When the Error-tolerant subgraph matching problem draws attention on graph G_1 (only all vertices of G_1 must be matched), error-tolerant graph matching is a problem that considers with equity to G_1 and G_2. To give the possibility to G_2 of capturing structural distortions, G_1' sets of vertices and edges should be extended. ε_{V_2} is a set of dummy vertices of G_1 such that $|\varepsilon_{V_2}| = |V_2|$ and $\mathcal{V}_1 = V_1 \cup \varepsilon_{V_2}$. Consequently, $\mathcal{V}_1$ and $\mathcal{V}_2$ have the same size (i.e, $N = |\mathcal{V}_1| = |\mathcal{V}_2|$). Similarly, ε_{E_2} is a set of dummy edges of G_1 and $\mathcal{E}_1 = E_1 \cup \varepsilon_{E_2}$.

Problem 4. *Error-Correcting Graph Matching (ECGM)*
A function $f : \mathcal{V}_1 \rightarrow \mathcal{V}_2$ *is an error-tolerant graph matching from* $G_1 = (\mathcal{V}_1,\mathcal{E}_1,\mu_1,\zeta_1)$ *to* $G_2 = (\mathcal{V}_2,\mathcal{E}_2,\mu_2,\zeta_2)$.
Function f must fulfill certain conditions:

1. $\forall u_i \in V_1 \Rightarrow f(u_i) \in \mathcal{V}_2$
2. $\forall v_i \in V_2 \Rightarrow f^{-1}(v_i) \in \mathcal{V}_1$
3. $\forall u_i, u_j \in V_1, (u_i, u_j) \in E_1 \Rightarrow (f(u_i), f(u_j)) \in \mathcal{E}_2$
4. $\forall v_i, v_j \in V_2, (v_i, v_j) \in E_2 \Rightarrow (f^{-1}(v_i), f^{-1}(v_j)) \in \mathcal{E}_1$
5. $\forall u_i \in \mathcal{V}_1, \mu_1(u_i) \approx \mu_2(f(u_i))$
6. $\forall (u_i, u_j) \in \mathcal{E}_1, \zeta_1((u_i, u_j)) \approx \zeta_2((f(u_i), f(u_j)))$

Mapping a vertex $u_i \in V_1$ to a dummy vertex if often called a deletion of u_i while mapping a dummy vertex to $v_i \in V_2$ is referred to an insertion operation. This vocabulary is inspired by the string edit distance. The string edit distance is a way of correcting a string to correspond to a second one by means of edit operations. Therefore this graph matching problem is often denoted as **error-correcting graph matching**.

2.2.2.4 Multivalent Matching

All the aforementioned matching problems, whether exact or error-tolerant ones, belong to the univalent family in the sense of allowing one vertex to be matched to at most one vertex in the other graph. In multivalent matching, a vertex can be matched to zero or many vertices [Sorlin et al., 2007].

Details and more graph matching problems are presented in Appendix A.

2.2.2.5 Error-tolerant matching cost

In error-tolerant GM, a measurement of the strength of matching vertices and/or edges is called *cost*. This cost is applicable on both graph structures and attributes. The basic idea is to assign a penalty cost to each matching operation according to the amount of distortion that it introduces in the transformation. When (sub)graphs differ in their attributes or structures, a high cost is added in the matching process. Such a cost prevents dissimilar (sub)graphs from being matched since they are different.

2.2.3 Graph Edit Distance

The graph edit distance (GED) was first reported in [Tsai et al., 1979, A. Sanfeliu, 1983, H. Bunke, 1983]. GED is a dissimilarity measure for graphs that represents the minimum-cost sequence of basic editing operations to transform a graph into another graph by means classically included operations: insertion, deletion and substitution of vertices and/or edges. Therefore, GED can be formally represented by the minimum cost edit path transforming one graph into another. Edge operations are taken into account in the matching process when substituting, deleting or inserting their adjacent vertices. From now on and for simplicity, we denote the substitution of two vertices u_i and v_k by $(u_i \rightarrow v_k)$, the deletion of vertex u_i by $(u_i \rightarrow \epsilon)$ and the insertion of vertex v_k by $(\epsilon \rightarrow v_k)$. Likewise for edges (u_i, u_j) and (v_k, v_z), $((u_i, u_j) \rightarrow (v_k, v_z))$ denotes edges substitution, $((u_i, u_j) \rightarrow \epsilon)$ and $(\epsilon \rightarrow (v_k, v_z))$ denote edges deletion and insertion, respectively.

An edit path (λ) is a set of edit operations o_i where $i = 1 \dots k$ and k is the number of edit operations. This set is referred to as *Edit Path* and it is defined in Definition 3.

Definition 3. *Edit Path*
A set $\lambda = \{o_1, \cdots, o_k\}$ of k edit operations o_i that transform G_1 completely into G_2 is called a (complete) edit path.

Let $c(o_i)$ be the cost function measuring the strength of an edit operation o_i. Let $\Gamma(G_1, G_2)$ be the set of all possible edit paths (λ). The graph edit distance problem is defined by Problem 5.

Problem 5. *Graph Edit Distance (GED)*
Let G_1 = (V_1,E_1,μ_1,ζ_1) and G_2 = (V_2,E_2,μ_2,ζ_2) be two graphs, the graph edit distance between G_1 and G_2 is defined as:

$$d_{min}(G_1, G_2) = \min_{\lambda \in \Gamma(G_1, G_2)} \sum_{o_i \in \lambda} c(o_i) \tag{2.1}$$

The GED problem is a minimization problem and d_{min} is the best distance. In its general form, the GED problem (Problem 5) is very versatile. The problem has to be refined to cope with the constraints of a graph matching problem. First, let us define constraints on edit operations (o_i) in Definition 4.

Definition 4. *Edit operations constraints*

1. *Deleting a vertex implies deleting all its incident edges.*
2. *Inserting an edge is possible only if the two vertices already exist or have been inserted.*
3. *Inserting an edge must not create more than one edge between two vertices.*

Second, let us define constraints on edit paths (λ) in Definition 5. This type of constraint prevents the edit path to be composed of an infinite number of edit operations.

Definition 5. *Edit path constraints*

1. *k is a finite positive integer.*
2. *A vertex/edge can have at most one edit operation applied on it.*

Finally, let us define the topology constrain in Definition 6. This type of constraints avoids edges to be matched without respect to their adjacent vertices.

Definition 6. *Topology constraints*

1. *The topology constraint implies that matching (substituting) two edges $(i, j) \in E_1$ and $(k, l) \in E_2$ is valid if and only if their incident vertices are matched ($(i \to k)$ and $(j \to l)$).*

An important property of the GED can be inferred from the topology constraint defined in Definition 6.

Property 1. *The edges matching are driven by the vertices matching*

1. *Assuming that constraint defined in Definition 6 is satisfied then three cases can appear :*

(a) *If there is an edge* $e_{ij} = (u_i, u_j) \in E_1$ *and an edge* $e_{kl} = (v_k, v_l) \in E_2$*, edges substitution between* (u_i, u_j) *and* (v_k, v_l) *is performed (i.e.,* $(e_{ij} \rightarrow e_{kl})$*).*

(b) *If there is an edge* $e_{ij} = (u_i, u_j) \in E_1$ *and there is no edge between* v_k *and* v_l *then an edge deletion of* (u_i, u_j) *is performed (i.e.,* $(e_{ij} \rightarrow \epsilon)$*).*

(c) *If there is no edge between* u_i *and* u_j *and there is an edge between and an edge* $e_{kl} = e(v_k, v_l) \in E_2$ *then an edge insertion of* $e(v_k, v_l)$ *is performed (i.e.,* $(\epsilon \rightarrow e_{kl})$*).*

The GED problem has been proved to be NP-hard by [Zeng et al., 2009]. So, unless $\mathcal{P} = \mathcal{NP}$, solving the problem to optimality cannot be done in polynomial time of the size of the input graphs. Regarding the complexity proof, [Zeng et al., 2009] have used a reduction of an induced subgraph isomorphism instance (see Problem 2.2.1.1 to a GED instance.

2.2.3.1 On the relation between GED and error-tolerant graph matching

The GED problem defined in Problem 5 and refined with constraints defined in Definitions 4, 5 and 6 is equivalent to the error-correcting graph matching problem defined in Problem 4 (ECGM). In this manuscript, the GED problem will always refer to the constrained version of the original problem.

2.2.3.2 Cost function : definition and discussion

Cost function $c(.)$ can be expressed thanks to the vertex/edge attributes. For instance, $c(u_i \rightarrow v_k) = Sub(\mu_1(u_i), \mu_2(v_k))$. The function Sub stands for substitution and explicitly exposes the labelling functions μ. If attributes are numeric, an example of the substitution function is the L2 norm $Sub = ||\mu_1(u_i) - \mu_2(v_k)||_2$. Neuhaus and Bunke [Neuhaus and Bunke., 2007] have shown that if each operation cost satisfies the criteria of a distance (positivity, uniqueness, symmetry, triangular inequality) then the edit distance defines a metric between graphs and it can be inferred that if $GED(G_1, G_2) = 0 \Leftrightarrow G_1 = G_2$.
Furthermore, it has been shown that standard concepts from graph theory, such as graph isomorphism, subgraph isomorphism, and maximum common subgraph, are special cases of error correcting graph matching under particular cost functions [Bunke, 1997, 1999]. Thus, any algorithm that implements error correcting graph matching can be used for the computation of graph isomorphism, subgraph isomorphism, and maximum common subgraph if it is run under an appropriate cost function. Conversely, for certain cost functions, algorithms for graph isomorphism, subgraph isomorphisms, or maximum common subgraph detection can be used to implement error correcting graph matching.
Another aspect to consider, when integrating the GED into a final application, is that cost functions must reflect the user need, thus they can be learned to fit a specific goal. For instance, the goal can be to reduce the gap between the ground-truth matchings and the optimal matchings.
Finally, it is worth to mentioned that graph matching difficulty can be reduced when the cost functions allow to easily differentiate between vertices and edges of the two graphs.

Now that we have defined graph matching problems, we can highlight how they are used in structural pattern recognition problems.

2.3 Structural pattern recognition problems

2.3.1 Graph-based search

Definition 7. *Graph-based search*

An unknown graph that models an object must be compared with all graphs in a database of known objects in order to find similarities. [Zeng et al., 2009] classify graph searches into three categories, for a database of graphs $\mathcal{D} = \{g_1, g_2, \cdots, g_n\}$ and a query graph q:

- *Full search: find all graphs g_i in $\mathcal{D}$ that are the same as q.*
- *Subgraph search: find all graphs g_i in $\mathcal{D}$ that contain or are contained by q.*
- *Similarity search: find all graphs g_i in $\mathcal{D}$ that are similar to q, based on some defined similarity measure.*

2.3.2 Graph classification

Definition 8. *Graph Classification*
Let $\mathcal{D}$ be the set of graphs and let $\mathcal{T}$ be the set of classes. Given a graph training set $TrS = \{(G_j, t_j)\}_{j=1}^{M}$, where $G_j \in \mathcal{D}$ is a graph and $t_j \in \mathcal{T}$ is the class of the graph. The classifier induces from TrS a mapping function $f : \mathcal{G} \to \mathcal{T}$ which assigns a class to an unknown graph from the test set TeS and $\mathcal{G}$ is the graph space.

Graph classifiers can be categorized into two categories whether the classifier operates in a graph space or in a vector space.

2.3.2.1 Vector space:

2.3.2.1.1 Explicit graph embedding ($\phi : \mathcal{G} \to \mathbb{R}^n$) A first one consists in transforming the initial structural problem in a common statistical pattern recognition one by describing the graphs with vectors in an Euclidean space. Such an approach can be achieved thanks to hand-crafted features extracted from the graphs [Luqman et al., 2013] or thanks to end-to-end learning methods as Graph Neural Networks [Nowak et al., 2017]. In such a context, some features (vertex degree, labels occurrence histograms,etc.) are extracted from the graph. Hence, the graph is projected in a Euclidean space and classical machine learning algorithms can be applied.

2.3.2.1.2 Implicit graph embedding ($k :< \mathcal{G}, \mathcal{G} >\to \mathbb{R}$) Another family of approaches also consists in using kernel-based machine learning algorithms. In the kernel approaches, an explicit data representation is of secondary interest. That is, rather than defining individual representations for each pattern or object, the data at hand is represented by pairwise comparisons only. The graphs are not explicitly but implicitly projected in a Euclidean space without defining the function ϕ. More formally, under given conditions, a similarity function can be replaced by a graph kernel function $k :< \mathcal{G}, \mathcal{G} >\to \mathbb{R}$. Most kernel methods can only process kernel values which are established by symmetric and positive definite kernel functions. Many kernels have been proposed in the literature [Gaüzère et al., 2012]. In most cases, the graph is embedded in a feature space composed of label sequences through a graph traversal. According to this traversal, the kernel value is then computed by measuring similarity between label sequences. In [Neuhaus and Bunke., 2007], graph kernels based on graph matching have been proposed to improve their expressiveness.

2.3.2.1.3 Dissimilarity space embedding Another possible approach also consists in projecting the graphs in a Euclidean space of a given dimension but using a distance matrix between each pairs of graphs. Each line of the distance matrix is a feature vector of a given graph. In such cases, a dissimilarity measure between graphs has to be designed [Bunke and Riesen, 2008]. Kernels can be derived from the distance matrix. It is the case for multidimensional scaling methods proposed in [Roth et al., 2003].

2.3.2.2 Graph space ($d : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$):

This paradigm is characterized by the fact that classification and learning problems are directly faced in the graph space, i.e. working on the graphs describing the objects at hand. The objects are classified by comparing the corresponding graphs, using suited matching algorithms. This paradigm operates directly on the graph space and can thus capture more structural distortions. Any distance-based classifier can be involved in the classification task.

2.3.2.3 Discussion

The recent rise of graph kernels and graph embedding methods might lead to state that the traditional gap between statistical and structural pattern recognition has been bridged. Yet, graph embedding crucially depends on similarity or dissimilarity computation on graphs. That is the topic of (efficient) graph comparison is still of high importance. We have defined a classifier as a high level function $f : \mathcal{G} \to \mathcal{T}$. Behind this black box view is hidden many general problems such as :

- Data reduction: Graph prototypes/clustering/indexing/partitioning [Musmanno and Ribeiro, 2016]
- Graph distance [Riesen, 2015]

The objectives of data reduction are (i) to overcome the well-known disadvantages of the large storage requirements, the large computational effort and the sensitivity to noisy examples and (ii) to keep classification performance as high as possible. The choice or the learning of the distance function is an important element in classification. The goal is to design fast and discriminant distance functions.

2.3.3 Graph analytic problems

Graph-based search and graph classification problems assume that a data set containing many graphs is available. On the other hand, graph analytic problems focus mainly on a single graph. Accordingly, **graph analytic problems do not belong to structural pattern recognition problems**.

Graph analytic tasks can be broadly abstracted into the following four categories: (a) vertex classification, (b) link prediction, (c) vertex clustering, and (d) visualization. A graph can be a friendship network, a Protein-Protein interaction network, a 3D mesh, or a Telecom network for instance. Vertex classification aims at determining the label of vertices based on other attributed vertices and the topology of the graph. Link prediction refers to the task of predicting missing links or links that are likely to occur in the future. Clustering is used to find subsets of similar vertices and group them together; finally, visualization helps in providing insights into the structure of the graph. Reviews of graph embedding methods for graph analytic problems can be found in theses two surveys [Goyal and Ferrara, 2017, Cai et al., 2017].

2.4 Applications and representations

2.4.1 Applications

Graph-based search (Definition 7) or classification (Definition 8) problems appear in many application fields and a taxonomy has been proposed by [Stauffer et al., 2017, Conte et al., 2004]. They

belong to many research fields such as Pattern Recognition, Computer Vision, Chem-informatics, Bio-informatics and Document analysis. The following examples are not exhaustive but they provide an interesting picture of the wide range of applications based on structural pattern recognition.

2.4.1.1 Image analysis.

Graphs are used in images to represent objects and patterns. They are flexible, so they can represent objects in both 2D- or 3D-images. Vertices model the main components that form the object, and each vertex has a list of attributes that characterizes the component, e.g. (x; y) - coordinates, color intensities around, special features, etc. Then, the edges are used to link the components, with additional attributes to carry information describing those links. Examples of graphs modeling objects, e.g. houses, cars, bikes and even human face features [Zhang et al., 2016b, Mateus et al., 2008]. Graphs can also describe silhouettes or skeletons of objects extracted from videos [Singh and Mohan, 2017, Jin Chang and Demiris, 2015]. Then, the graph matching problems can be solved in order to compare objects and patterns and therefore to perform: object detection and recognition, image segmentation [Yu and Wang, 2016]. Visual question answering (VQA) is another interesting problem in image analysis where a question is formulated in natural language about the content of an image. With structured representations of both scene contents and questions, graph-based methods operate over graphs of the scene objects and over the question words [Teney et al., 2017].

2.4.1.2 Handwritten document analysis.

Graphs are constructed over segmented words in images, where vertices represent the keypoints or strokes, and the edges link pairs of keypoints or strokes. A graph models the documents words and their relations. Then, graph matching can be applied between a query graph and documents graphs to find correspondences. Such an application is called keyword spotting and there exists many works in the literature that use graph matching [Stauffer et al., 2018].

2.4.1.3 Biometrics.

Retina vessels, fingerprints or signatures are considered as biometrical characteristics. There are many applications with the goal of identifying an individual based on the fingerprint or signature. So, graphs can be used to model a fingerprint, where vertices represents segmented core areas, and edges relates adjacent areas. Graphs can then be classified [Choi and Kim, 2010].

2.4.1.4 Bio- and Chem-informatics.

In the field of Bio-informatics, graphs are used to model DNA, protein sequences and enzymes. This enables analyzing biological structures. A very important example is the ability of detecting cancerous tissues. Tissues are modeled by graphs and then a classifier is built to classify normal, low-grade and high-grade cancerous tissues [Ozdemir and Gunduz-Demir, 2013]. In chemistry field and precisely when considering chemical molecules, graphs form a natural representation of the atom-bond structure of molecules. Each vertex of the graph then represents an atom, while an edge represents a molecular bond [Raymond and Willett, 2002]. By using graph matching, it provides a way to compare molecules between each other and to detect similar activities and properties, which answers a major question in this field.

2.4.1.5 Malware detection.

The efficiency of using graphs to construct relational models for malicious executables of the same family, makes it suitable to employ graph matching in the task of malware detection in Anti-viruses. Graphs are called *call graphs* and represent a malware sample with certain variations. Then, based on graph matching solutions after comparing the graphs, certain properties can be extracted based on the similarities found [Bourquin et al., 2013, Ahmed et al., 2012]. In those examples, the graph matching problem is not the main problem, but it is used for graphs comparison and then builds up on it to achieve the objectives of detecting malicious executables.

2.4.2 Graph-based representations

While describing the applications, we have also depicted some graph-based modelling, we now propose to organize them. Graph-based representations can be split into two parts whether raw data is structured or not.

2.4.2.1 Native representations

A native representation appears when data are naturally or explicitly expressed into graphs. Structured or relational data fall in this category. Prominent examples of classes of patterns, which can be formally represented in a more suitable and natural way by means of graphs rather than with feature vectors, are chemical compounds, digital-born documents, and networks.

2.4.2.2 Graph constructed from non-relational data

Built-based representations are graph-based representation built on top of unstructured data such as Euclidean data (Audio signals, images). At the low level, such data can be represented by a grid. For example, an image can be represented by a 2D-grid where each pixel is a vertex and each vertex is connected to its 4 or 8 neighbours. In image analysis, graphs can be built from higher levels such as skeletons, silhouettes, irregular partitions of regions.

2.4.3 Why pattern recognition based on graphs ?

In this section, we give an insight about the main motivations to use graph-based pattern recognition methods.

1. *Detect and recognize at the same time.* Object detection is one of the problems of image analysis. This type of problem is more difficult than the recognition of isolated objects as it is necessary to simultaneously segment and recognize the object. Subgraph maching can be helpful to solve at once the detection and recognition problems. Subgraph isomorhpism can be employed to spot where a subgraph pattern is located within a larger graph.

2. *Graphs by nature.* Relational data or structured data are designed as graphs. Non-vectorial methods can guarantee to preserve the topological information.

3. *Combining sources of data.* A graph can help at combining different sources of information. For instance, an image can be merged with a knowledge graph to bring semantic to an image annotation procedure. Reversely, a graph of an image can be mixed with a question graph to solve the visual question answering problem [Teney et al., 2017, Lee et al., 2018, Marino et al., 2017].

4. *Beyond Euclidean data.* Considering Euclidean data like images, graphs can be used to develop non-local approaches and to go beyond the standard 8x8 connectivity. While a pixel is linked to its 8 neighbours in a image, in a graph it is possible to be **non-local** and to extend the neighborhood definition. A pixel can be connected to every pixel in the image and each relation can be enriched by a set of features.

5. *Matching matter.* Graphs are crucial when the matching between components of the two graphs must be analysis to interpreter the results. For instance, when parts of an object must be tracked to understand the object behaviour.

6. *Relation matter.* Finally, graphs are essential when relationship between components is fundamental and relationship brings sense to the data. The data are characterized by complex structural relationships rather than the statistical distribution of a fixed set of features.

Graph-based representations are of pivotal importance in computer vision, pattern recognition and machine learning. Graph representations also pose unique problems in machine learning, since they are non-vectorial in nature and require new methodology to be developed. For these reasons the design of efficient graph-based algorithms for pattern recognition will certainly be one of the major challenges over the next decades.

2.5 Surveys and organization

The use of graphs in Pattern Recognition (PR) dates back to the early 70s [Fischler and Elschlager, 1973, Ullmann, 1976]; Good surveys of graph based techniques have been published up to now on different areas: graph-based representations, graph matching, graph edit distance, graph embedding and graph kernels [Hancock and Wilson, 2012, Gao et al., 2010, Bunke and Riesen, 2012, Wilson and Zhu, 2008, Conte et al., 2004, Riesen and Bunke, 2010b, Riesen, 2015, Vento, 2015] provide an extensive overview of the literature over the last 40 years by introducing a detailed categorization of graph-based methods.

In this dissertation, we propose to organize the literature on graph matching and graph-based search in two parts. The first part (Section 3) is dedicated to learning-free proposals where methods

concentrate their effort on the solution of optimization problems. In the second part (Section 4), a learning phase is required by the approaches and machine learning techniques are of first importance.

Chapter 3

Combinatorial optimization for graph matching and graph classification

Contents

3.1 Graph matching . . . 24
3.1.1 State of the art . . . 25
3.1.1.1 Related problems . . . 25
3.1.1.1.1 Quadratic Assignment Problem (QAP) . . . 25
3.1.1.1.2 Maximum a posteriori (MAP)-inference problem for Conditional Random Field (CRF) . . . 26
3.1.1.1.3 Summary of the related problems . . . 27
3.1.1.2 Related models . . . 27
3.1.1.3 Related methods . . . 30
3.1.1.3.1 Exact methods . . . 30
3.1.1.3.2 Heuristics methods . . . 31
3.1.1.4 Summary . . . 36
3.1.1.5 Discussions on the literature of graph matching . . . 36
3.1.1.5.1 Model GMIQP*vs* Model SGMIQP . . . 38
3.1.1.5.2 Performance evaluation metric . . . 38
3.1.1.5.3 Data sets . . . 41
3.1.1.5.4 Graph matching library . . . 43
3.1.1.5.5 Discussion on the running times . . . 44
3.1.1.5.6 Effectiveness analysis . . . 44
3.1.1.5.7 Reasoning about effectiveness and speed-up . . . 45
3.1.2 Open problems . . . 46
3.1.3 Contribution . . . 47
3.1.3.1 Deadlock 1: models and algorithms for exact methods . . . 47
3.1.3.1.1 Motivation . . . 47
3.1.3.1.2 Exact methods: mathematical models . . . 47
3.1.3.1.3 Exact method: Branch and Bound algorithm (DF) . . . 51
3.1.3.2 Deadlock 2: Performance evaluation of graph matching methods 51
3.1.3.2.1 Motivation . . . 51
3.1.3.2.2 Graph data repository for error tolerant graph matching 52
3.1.3.2.3 ICPR 2016 - graph edit distance contest (GDC) . . . 52

3.1.3.3 Deadlock 3: heuristics performance according to speed and effectiveness criteria . . . 54
3.1.3.3.1 Heuristics-based on a mathematical model . . . 54
3.1.3.3.2 Anytime Branch and Bound (ADF) . . . 55
3.1.3.4 Deadlock 4: Relation between Problem ETSGM and Problem ECGM . . . 57
3.1.4 Summary . . . 61
3.2 Graph classification . . . 62
3.2.1 State of the art . . . 62
3.2.1.1 Problem definition . . . 62
3.2.1.2 kNN methods . . . 63
3.2.1.2.1 Exact methods . . . 63
3.2.1.2.2 Heuristic methods . . . 64
3.2.1.3 Analysis . . . 65
3.2.2 Open problems . . . 67
3.2.3 Contribution . . . 67
3.2.3.1 Deadlock 5: What is the impact of GED heuristics on the kNN problem? . . . 67
3.2.3.2 Deadlock 6: Is there a way to specialize a line search method to operate in graph space? . . . 69
3.2.3.2.1 Motivation . . . 69
3.2.3.2.2 MGED problem . . . 71
3.2.3.2.3 One-tree depth first algorithm to solve the kMGED problem . . . 72
3.2.3.2.4 Used GED solver . . . 72
3.2.3.2.5 Complexity and time constraint . . . 73
3.2.3.2.6 Theoretical discussion around the impacts of the parameters . . . 73
3.2.3.3 Deadlock 7: What is the impact of GED heuristics in a classification context ? . . . 74
3.2.3.3.1 1) Are heuristics sufficiently accurate for classification tasks when cost functions are chosen based on expert knowledge? . . . 74
3.2.3.3.2 Why heuristics can provide good classification results? 75
3.2.3.3.3 Question 2: Are heuristics sufficiently accurate for classification tasks when cost functions are learned? . 76
3.2.4 Summary . . . 78

This section is split into two parts graph matching (Section 3.1) and graph classification (Section 3.2). Each part is then broken down in three steps. First, the state of the art is summed up. Second, deadlocks and open problems are expressed. Finally, contributions are presented. References about our work are given at the end of each section 3.1 and 3.2 through a quick summary. Note that the state of the arts do not include our work. This is intentionally done to highlight how our contributions help to lift the deadlocks.

3.1 Graph matching

In this section, we focus on error-tolerant graph matching methods. Such methods are more convenient for pattern recognition. In reality, graphs suffer from the presence of both noise and

distortions due to the graph extraction process or due to the presence of noise in the raw data. Thus, exact graph matching problems fail to answer whether two graphs G_1 and G_2 are similar or not.

3.1.1 State of the art

The sate of the art is split into 4 parts. First, graph matching problems are linked to other fundamental problems in Operational Research and Machine Learning. Second, graph matching problems are expressed in terms of mathematical models. Third, the main solving methods are explained. Finally, an analysis is given.

3.1.1.1 Related problems

The aim of this section is to show the equivalence between the graph matching problems and other important problems from the communities of Operational Research and Machine Learning.

3.1.1.1.1 Quadratic Assignment Problem (QAP) The QAP was introduced by Koopmans and Beckmann in 1955 [Koopmans and Beckmann, 1955] and extended by [Lawler, 1963] as a mathematical problem for the location of a set of indivisible economical activities. Consider the problem of allocating a set of facilities to a set of locations, with the cost being a function of the distance and flow between the facilities, plus costs associated with a facility being placed at a certain location. The objective is to assign each facility to a location such that the total cost is minimized. QAP is a very important problem and covers a large range of applications: Bandwith minimization of a graph, Economics, Molecular conformations in chemistry, Scheduling, Supply Chains, Manufacturing lines, ...

The formal definition of the quadratic assignment problem is as follows:

Problem 6. *Quadratic Assignment Problem (QAP)*
Specifically, we are given three $N \times N$ input matrices with real elements F, C and B, where F_{ij} is the flow between the facility i and facility j, C_{kl} is the distance between the location k and location l, and B_{ik} is the cost of placing facility i at location k. The Koopmans-Beckmann version of the QAP can be formulated as follows: Let N be the number of facilities and locations and denote by pe the set $pe = \{1, 2, ..., N\}$.

$$\min_{\phi \in S_n} \sum_{i=1}^{N} \sum_{j=1}^{N} F_{ij} \cdot C_{\phi(i)\phi(j)} + \sum_{i=1}^{N} B_{i\phi(i)}$$

where S_n is the set of all permutations and ϕ is a function $\phi : pe \to pe$.

The QAP is known to be $\mathcal{NP}$-hard [Sahni and Gonzalez, 1976].

To reflect the error-tolerant graph matching problem, the matrices F, C and B must be redefined along with the role of the function ϕ. $\phi(i)$ represents an assignment of i with k, $\forall i \in \mathcal{V}_1$ and $\forall k \in \mathcal{V}_2$. $B_{i\phi(i)}$ must denote the vertex matching cost between vertices i and k. Similarly, $\phi(i)\phi(j)$ is the matching of an edge (i, j) with an edge (k, l), $\forall (i, j) \in \mathcal{V}_1 \times \mathcal{V}_1$ and $\forall (k, l) \in \mathcal{V}_2 \times \mathcal{V}_2$. By setting $D_{i,k,j,l} = f_{ij}C_{kl}$, $D_{i\phi(i),j\phi(j)}$ is the cost to match (i, j) with (k, l).

Problem 7. *Error-tolerant graph matching as a QAP (QAPGM)*

$$\min_{\phi \in S_n} \sum_{i=1}^{N} \sum_{j=1}^{N} D_{i\phi(i),j\phi(j)} + \sum_{i=1}^{N} B_{i\phi(i)}$$

where S_n is the set of all permutations and ϕ is a function $\phi : pe \to pe$.

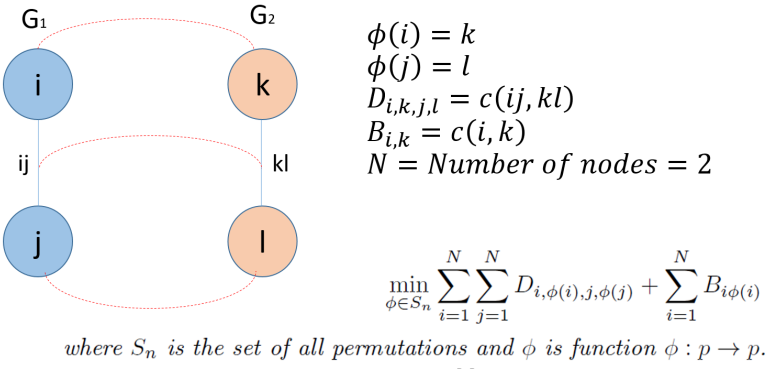


Figure 3.1: An example of graph matching modelled as a QAP.

Under this formalism, the error-correcting graph matching problem (Problem 4) and the QAP (Problem 7) are equivalent (see [Riesen, 2015] for more details).

An example is depicted in Figure 3.1.

Now let us take a look to another interesting problem called Maximum *a posteriori*(MAP)-inference problem for Conditional Random Field (CRF) and see how it is related to graph matching.

3.1.1.1.2 Maximum a posteriori (MAP)-inference problem for Conditional Random Field (CRF) Finding the most likely configuration of a Conditional Random Field (CRF), also called MAP-inference or energy minimization problem for graphical models, is of big importance in computer vision, bioinformatics, communication theory, statistical physics, combinatorial optimization, signal processing, information retrieval and statistical machine learning.

Definition 9. *Conditional random fields (CRF)*
Let $G = (V, E)$ be an undirected graph. For each node $i \in V$, a variable y_i is associated to i. y_i takes its values in a finite set of labels $Y_i \subset \{(1, 0, ..., 0), (0, 1, 0, ..., 0), ..., (0, ..., 0, 1)\}$. Hence, each label corresponds to a unit vector. Notation, $\mathcal{Y}_V$ denotes the Cartesian product $\Pi_{i \in V} Y_i$. A vector $y \in \mathcal{Y}_V$ with coordinates $(y_i, \cdots, y_j)$ $\forall i, j \in V$ is called a labeling. Likewise, $Y_{ij} = Y_i \times Y_j$ indicates the label set of an edge (i, j). Functions $\theta_i : Y_i \to \mathbb{R}$ $\forall i \in V$, and $\theta_{ij} : Y_{ij} \to \mathbb{R}$ $\forall (i, j) \in E$, are the unary and pairwise potentials associated with the nodes and edges of G which define a local quality of labels and label pairs.

Problem 8. *MAP-inference problem (MAPCRF)*
$\min_{x \in \mathcal{Y}_V} \sum_{i \in V} \theta_i(y_i) + \sum_{ij \in E} \theta_{ij}(y_{ij})$

The MAP-inference problem is the problem of finding the minimum cost labeling. The MAP-inference problem is known to be $\mathcal{NP}$-hard [Lafferty et al., 2001]. To fit to the error-correcting graph matching problem (Problem ECGM), V must be equal to $\mathcal{V}_1$. The label y_i must be equal to a possible matching between $i \in \mathcal{V}_1$ and $k \in \mathcal{V}_2$. This matching can be encoded by a one-hot vector to ensure that each label corresponds to a unit vector. For instance, $y_i = (0, 0, 1, 0)$ represents the matching of node i with the third node in $\mathcal{V}_2$. By extension, Y_i represents for node i all the possible matching: $(i \to k) \forall k \in \mathcal{V}_2$. $\mathcal{Y}_{\mathcal{V}_1}$ denotes the Cartesian product $\Pi_{i \in \mathcal{V}_1} Y_i$. The Problem MAPCRF must be refined because in the graph matching problem no label can be taken twice.

Let a common universe $\mathcal{L}$ of labels be given such that $Y_i = \mathcal{L}$ $\forall i \in \mathcal{V}_1$. We require each label $la \in \mathcal{L}$ to be taken only once, i.e. $|\{i \in \mathcal{V}_1 | y_i = la\}| = 1$. In other words, the problem is to find a mapping $\mathcal{V}_1 \to \mathcal{L}$. This problem can be stated as

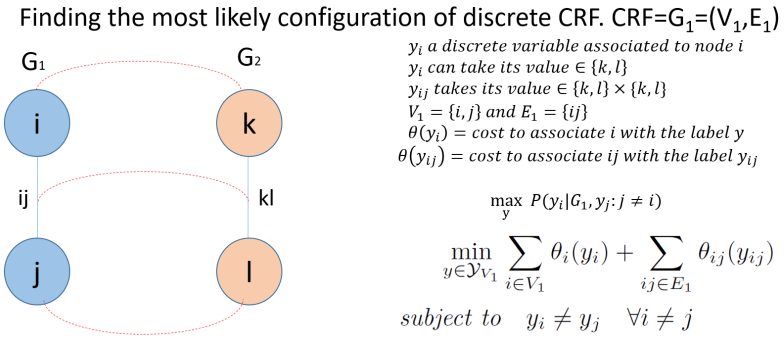


Figure 3.2: An example of graph matching modelled as a CRF.

Problem 9. *Error-tolerant graph matching as a MAP-inference problem (MAPCRFGM)*

$$\min_{y \in \mathcal{Y}_{\mathcal{V}_1}} \sum_{i \in \mathcal{V}_1} \theta_i(y_i) + \sum_{ij \in \mathcal{E}_1} \theta_{ij}(y_{ij})$$

$$\textit{subject to} \quad y_i \neq y_j \quad \forall i \neq j$$

This problem transformation has been slightly discussed in [Swoboda et al., 2017] but here we have presented a more concrete relation between both problems.

An example is depicted in Figure 3.2.

3.1.1.1.3 Summary of the related problems

In general, the error-tolerant graph matching problems are equivalent to the QAP without any modification. The error-tolerant graph matching problems are equivalent to a modified version of the MAP-inference problem. The error-tolerant graph matching problems are equivalent to a constrained version of the GED problem. The relation between problems are pictured out in Figure 3.3. Positioning the graph matching problems with respect to other fundamental problems is important because QAP or MAP-inference problems are well studied. QAP or MAP-inference solvers could be useful to solve the graph matching problems.

3.1.1.2 Related models

In Section 2.2, mathematical descriptions of graph matching problems were given.

From these descriptions, mathematical models can be expressed. A mathematical model is composed of variables, constraints and an objective functions. From this rigorous formulation, no ambiguity is left behind but a single problem can be expressed by many different models. An Integer Quadratic Programm (IQP) is a model with a quadratic objective function of the variables and linear constraints of the variables.

A graph matching solution is defined as a subset of possible correspondences $y \subset \mathcal{V}_1 \times \mathcal{V}_2$, which are represented by a binary assignment matrix $Y \in \{0,1\}^{N \times N}$, where N denotes the size of $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively. If $u_i \in \mathcal{V}_1$ matches with $v_k \in \mathcal{V}_2$, then $Y_{i,k} = 1$, or $Y_{i,k} = 0$ otherwise. We denote by $y \in \{0,1\}^{N^2}$, a column-wise vectorized replica of Y. With this notation, the error-correcting graph matching problem (Problem ECGM) can be expressed as the problem of finding the assignment vector y^* that minimizes a score function $d(G_1, G_2, y)$ as follows:

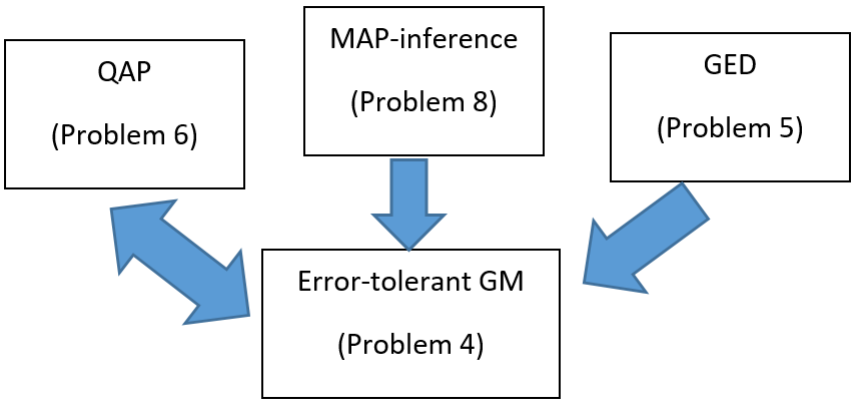


Figure 3.3: A relation between the Graph Edit Distance (GED), Quadratic Assignment (QAP), Maximum-a posteriori (MAP) inference and error-tolerant graph matching (ETSGM and ECGM) problems.

Model 1. *Error-correcting graph matching model : Integer Quadratic Program (GMIQP)*

$$y^* = \underset{y}{\text{argmin}} \quad d(G_1, G_2, y) \tag{3.1a}$$

$$\textit{subject to} \quad y \in \{0,1\}^{N \cdot N} \tag{3.1b}$$

$$\sum_{i=1}^{N} y_{ik} = 1 \quad \forall k \in [1, \cdots, N] \tag{3.1c}$$

$$\sum_{k=1}^{N} y_{ik} = 1 \quad \forall i \in [1, \cdots, N] \tag{3.1d}$$

Where $G_1 = (\mathcal{V}_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (\mathcal{V}_2, E_2, \mu_2, \zeta_2)$ are two graphs. $N = |\mathcal{V}_1| = |\mathcal{V}_2|$. Constraints 3.1c and 3.1d indicate that each vertex of a graph must be matched with only one vertex of the other graph. The function $d(G_1, G_2, y)$ measures the dissimilarity of graph attributes, and is typically decomposed into a first order dissimilarity function $c(u_i \to v_k)$ for a node pair $u_i \in \mathcal{V}_1$ and $v_k \in \mathcal{V}_2$, and a second-order similarity function $c(e_{ij} \to e_{kl})$ for an edge pair $e_{ij} \in \mathcal{V}_1 \times \mathcal{V}_1$ and $e_{kl} \in \mathcal{V}_2 \times \mathcal{V}_2$. Dissimilarity functions are usually represented by a symmetric dissimilarity matrix $D \in \mathbb{R}^{N^2 \times N^2}$ with $N = |\mathcal{V}_1| = |\mathcal{V}_2|$. A non-diagonal element $D_{ik,jl} = c(e_{ij} \to e_{kl})$ contains the edge dissimilarity and a diagonal term $D_{ik,ik} = c(u_i \to v_k)$ represents the vertex dissimilarity. Thus, the objective function of graph matching is defined as:

$$\begin{aligned} d(G_1, G_2, y) =& \sum_{i=1}^{N}\sum_{k=1}^{N} c(u_i \to u_k) \cdot y_{ik} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N} c(e_{ij} \to e_{kl}) \cdot y_{ik} \cdot y_{jl} \\ =& \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N} D_{ik,jl} \cdot y_{ik} \cdot y_{jl} \\ =& y^T D y \end{aligned} \tag{3.2}$$

In essence, the score accumulates all the dissimilarity values that are relevant to the assignment. Remember that a vertex can be matched to only one other vertex so $D_{ik,jl} = cst$ a large constant

value when $i = j$ and $k \neq l$. In addition, matching two not existing edges in E_1 and E_2 should not lead to any cost so if $(i, j) \notin E_1$ and $(k, l) \notin E_2$ then $D_{ik,jl} = 0$. More details about the model can be found in [Bougleux et al., 2017a].

Model 1(GMIQP) models the error-correcting graph matching problem. Other quadratic objective functions does exist in the literature [Lyzinski et al., 2016]:

$$tr(A_1 Y^T A_2 Y) \tag{3.3}$$

and

$$\|A_1 - Y^T A_2 Y\|^2 = \|A_1 Y - Y A_2\|^2 \tag{3.4}$$

where $tr(.)$ is the trace of a matrix. A_1, $A_2 \in \mathbb{R}^{N \times N}$ are the weighted adjacency matrices of the graphs and $Y \in \{0,1\}^{N \times N}$ is a permutation matrix. In equation 3.4, the problem consists in determining the permutation matrix minimizing the Frobenius norm of the difference between weighted adjacency matrix of the input graph and the permuted adjacency matrix of the target one.

Equations 3.4 and 3.3 only rely on weighted adjacency matrices and cannot deal with richly attributed graphs.

Different IQP models exist in the literature to model different problems such as subgraph matching (Problem 3 (ETSGM)) and maximum common sugraph (Problem 2 (MCS)) [Cho et al., 2014]. In particular, theses models define $K \in \mathbb{R}^{|V_1|.|V_2| \times |V_1|.|V_2|}$, it is called an affinity matrix or compatibility matrix. K is similar to D but stores similarities instead of costs or dissimilarities. In this representation, $K_{ij,kl} = 0$ means an impossible matching or a very dissimilar matching. The problem becomes a maximization problem as follows :

Model 2. *Error-tolerant subgraph matching model : Integer Quadratic Program (SGMIQP)*

$$y^* = \underset{y}{\text{argmax}} \quad y^T K y \tag{3.5a}$$

$$\textit{subject to} \quad y \in \{0,1\}^{|V_1| \cdot |V_2|} \tag{3.5b}$$

$$\sum_{i=1}^{|V_1|} y_{ik} \leq 1 \quad \forall k \in [1, \cdots, |V_2|] \tag{3.5c}$$

$$\sum_{k=1}^{|V_2|} y_{ik} \leq 1 \quad \forall i \in [1, \cdots, |V_1|] \tag{3.5d}$$

A part from IQP models, very different models of the error-tolerant graph matching problem can be observed in the literature. An Integer Linear Program (ILP) formulation of GED was proposed in [Justice and Hero, 2006]. An ILP is a mathematical model where the objective function is a linear combination of the variables. The objective function is constrained by linear combinations of the variables.

Model 3. *Error-correcting graph matching model : Integer Linear Program (ECGMILP)*

$$d(G_1, G_2) = \min_{Y,S,T \in \{0,1\}^{NXN}} \sum_{i=1}^{N} \sum_{k=1}^{N} \{c(u_i \to v_k) Y_{i,k} + \frac{\mathcal{K}}{2}(S_{i,k} + T_{i,k})\} \tag{3.6a}$$

$$s.t. \quad \sum_{j=1}^{N} A1_{i,j} \cdot Y_{j,k} - \sum_{l=1}^{N} A2_{l,k} \cdot Y_{i,l} + S_{i,k} - T_{i,k} = 0 \qquad \forall i,k \in [1,2,\cdots,N] \tag{3.6b}$$

$$\sum_{i=1}^{N} Y_{i,m} = \sum_{k=1}^{N} Y_{m,k} = 1 \qquad \forall m \in [1,2,\cdots,N] \tag{3.6c}$$

where Y is a permutation matrix representing all possible permutations of the vertices. Two matrices, S and T, are introduced (inspired by [Almohamad and Duffuaa, 1993]) to manage edges matching. $A\{n\} \in \{0,1\}^{NXN}$ is the modified adjacency matrix corresponding to G_n (see [Justice and Hero, 2006] for modifications). $\mathcal{K}$ is a constant cost for edges insertions and deletions. Two types of constraints are defined. Constraint 3.6b makes sure that when matching two couples of vertices, the edges between them have to be matched as well. Constraint 3.6c states that one vertex of G_1 (i.e., vertex) must be permuted with exactly one vertex of G_2.

This model has a limitation. It does not consider the attributes on edges, so edge substitution cost is 0 while deletion and insertion have a $\mathcal{K}$ fixed cost.

This is only one model to give some intuition but more ILP models can be found [Bodic et al., 2012].

QAP and MAP-inference problems hold their own IQP and ILP models. These models could be useful to model the error-tolerant graph matching problems too.

3.1.1.3 Related methods

Methods can be divided into two categories: exact methods and heuristic methods. An exact method computes an optimal solution of a given problem. On the opposite, heuristic methods compute sub-optimal solutions.

3.1.1.3.1 Exact methods For the family of error-tolerant graph matching problems, the run-time complexity of exact methods is not polynomial but exponential with respect to the number of vertices of the graphs.

3.1.1.3.1.1 Tree-based methods Pioneer approaches of graph matching algorithms are based on tree search [Tsai et al., 1979, Shapiro and Haralick, 1981]. In tree-based methods proceed to an implicit enumeration of all possible solutions without explicitly evaluating all of them by means of an ordered tree. It is constructed dynamically at run time by iteratively creating successor tree nodes. A tree node p here corresponds to a partial matching. At each iteration, the choice of the next tree node to be expanded is important. There are many strategies such that:

1. Depth-first: The most promising tree node that is a child of p is chosen.
2. Breadth-first: The most promising tree node that is at the same level of p is chosen.
3. Best-first: The most promising tree node is chosen without constraints (at the same level or not).

These search strategies require having a function $g(p)+h(p)$ to compute heuristically an estimation of the cost of exploring a given node further. $g(p)$ represents the cost of the partial matching accumulated so far whereas $h(p)$ denotes the estimated cost from p to a leaf node representing a complete solution. The sum $g(p) + h(p)$ is referred to as a lower bound $lb(p)$ [Fischer et al.,

2017]. Given that the estimation of the future costs $h(p)$ is lower than, or equal to, the real costs, an optimal matching from the root node to a leaf node is guaranteed to be found. Leaf nodes correspond to feasible solutions. In the simplest scenario, the estimation of the lower bound $h(p)$ of the future costs for the current node p is set to zero for all p. In the other extreme $h(p)$ would return the exact future costs in exponential time complexity which is unreasonable, of course. A good heuristic ($h(p)$) helps at pruning unfruitful branches. This family of methods belongs to the Branch and Bound methods.

3.1.1.3.1.2 ILP and IQP solvers In general, ILP and IQP formulations are solved by black-box solvers such as CPLEX, Gurobi, etc. These solvers are equipped with an arsenal of effective algorithms. Theses algorithms can be applied at two different moments: 1) a preprocessing stage before searching for a solution and 2) during the solving. For instance, let us mention two preprocessings that are embedded into the solvers. 1) Automatic cuts are sets of constraints added to the model for a given instance. Cuts are expected to reduce the search space (Gomory cuts, Disjunctive cuts, ...). 2) Preprocessing consists in analyzing an instance and figures out if some variables can be fixed to 0 or 1 in the graph matching model (with respect to Driebek penalty for instance). The aim is to reduce the number of variables to be fed to solving methods. A model is usually solved by a tree-based methods as mentioned in the prior paragraph. However, these algorithms differ by taking advantage of the mathematical formulation. During the search, lower bounds ($lb(p) = g(p) + h(p)$) are computed by continuous relaxation and problem decomposition for instance. Upper bounds can be computed by fast local searches. Cuts can be added to the model (Branch and Cut [Gomory, 1958]). Generally speaking, ILP are better solved than IQP by black-box solvers.Graph matching problems modeled as an ILP and solved by a black box solver can be found in [Justice and Hero, 2006] and [Bodic et al., 2012].

3.1.1.3.2 Heuristics methods Heuristics methods can be grouped into two families: problem reformulation and heuristic optimization. In the problem reformulation paradigm, the GM problem is reduced into a simpler problem. However, the optimal solution of the simpler problem is not the optimal solution of the original problem. In the heuristic optimization category, the GM problem is solved by an heuristic algorithms that only explore sub-parts of the solution space and thus leads to find near-optimal solutions. Each heuristic method can also be divided between deterministic and non-deterministic. The characteristic of deterministic strategies is that under the same conditions the same solution is always obtained.

3.1.1.3.2.1 Tree-based methods Heuristic methods can be derived from exact tree-based methods by truncating the search tree or over-estimating $h(p)$. Truncation of the search tree can be achieved by limiting the number of tree nodes (p) in memory (BeamSearch, PathLength, ... see [Neuhaus et al., 2006]) or by limiting the solution time.

3.1.1.3.2.2 QAP-based method As presented in Model GMIQP, graph matching problems can be modeled by a QAP (Problem 7). A number of problems such as traveling salesman and graph partitioning can be straightforwardly reduced to QAP. Due to its generality and flexibility, many solver paradigms were put to the test for QAP. These include, but are not limited to, convex relaxations based on Lagrangean decompositions [Karisch and Rendl, 1995], linear [Hahn and Grant, 1998], convex quadratic [Anstreicher and Brixius, 2001] and semi-definite Zhao et al. [1998] relaxations, which can be used either directly to obtain approximate solutions or just to provide lower bounds. Graph matching modeled as a QAP has been favored in recent graph matching researches. Many efficient approximate algorithms have been applied to graph matching instances : Graduated Assignment (GA)[Gold and Rangarajan, 1996], Spectral Matching (SM) [Leordeanu

and Hebert, 2005], Spectral Matching with Affine Constraint (SMAC) [Cour et al., 2007], Integer Projected Fixed Point (IPFP) [Leordeanu et al., 2009], Reweighted Random Walks Matching (RRWM) [Cho et al., 2010b], and Max-Pooling Matching (MPM) [Cho et al., 2014].

3.1.1.3.2.3 Continuous relaxation As presented in Models GMIQP and GMILP, error-tolerant graph matching problems are discrete optimization problems ($y \in \{0,1\}^{|\mathcal{V}_1|\cdot|\mathcal{V}_2|}$). A way to simplify the problem is to relax discrete variables to obtain continuous variables (for instance $y \in [0,1]^{|\mathcal{V}_1|\cdot|\mathcal{V}_2|}$). The continuous version of an ILP model is referred as an Linear Program (LP). A LP can be solved optimally in polynomial time by the interior points method [Potra and Wright, 2000] or efficiently solved by the simplex method.

The graph matching problems can be equivalent to the $\mathcal{NP}$-hard QAP. In general, the relaxed QAP is also a $\mathcal{NP}$-hard problem [Lyzinski et al., 2016]. If the dissimilarity matrix (D) is symmetric and negative definite then the relaxed QAP is convex and can be solved in polynomial time [Burkard et al., 1999, Lyzinski et al., 2016]. Nonetheless, the QAP relaxation can be efficiently approximately solved with Frank-Wolfe (F-W) methodology [Frank and Wolfe].

3.1.1.3.2.4 Franck-Wolfe based methods The Frank–Wolfe algorithm is an iterative first-order optimization algorithm for constrained convex optimization. In each iteration, the Frank–Wolfe algorithm considers a linear approximation of the objective function (given by the first order Taylor expansion), and moves towards a minimizer of this linear function. It is similar to the gradient descent algorithm but at each iteration, a linear program is solved. Franck-Wolfe algorithm is a relaxed QAP solver and the methods Path-following [Zaslavskiy et al., 2009] and Factorized Graph Matching (FGM) [Zhou and la Torre, 2016] rely on the expensive Franck-Wolfe algorithms.

3.1.1.3.2.5 Linear Sum Assignment Problem (LSAP) The Model GMIQP can be turned into a LSAP model, if $D_{ik,jl} = 0$ when $ik \neq jl \ \forall i,j \in \mathcal{V}_1$ and $k,l \in \mathcal{V}_2$. The linear assignment problem (LSAP) is exactly solvable in worst-case cubic time by the Hungarian method [Kuhn and Yaw, 1955]. In such a scenario, $D_{ik,ik}$ can be enriched to take into account the local neighbours of vertices i, k [Riesen and Bunke, 2009, Serratosa, 2015, Raveaux et al., 2010]. In [Riesen and Bunke, 2009], the memory requirements and execution times of this method are respectively proportional to $(n_1+n_2)^2$ and $(n_1+n_2)^3$ where n_1 and n_2 are the sizes of vertex sets ($n_1 = |V_1|$ and $n_2 = |V_2|$). In [Bougleux et al., 2017b], for the same results, the algorithm requires $O(n_1 n_2)$ memory space and $O(\min(n_1,n_2)^2 \max(n_1,n_2))$ execution times. In [Serratosa, 2015], a reduction of the the execution times $O(\min(n_1,n_2)^3)$ is achieved but at the price of an approximated solution of the LSAP problem.

3.1.1.3.2.6 Integer Projected Fixed Point (IPFP) In [Leordeanu et al., 2009], the Franck-Wolfe methodology is adapted to solve the discrete QAP. The method is named Integer Projected Fixed Point (IPFP). It is an algorithm initially proposed to find a solution to the quadratic assignment problem in the context of graph matching and MAP- inference problems. The algorithm tries to find a solution to both relaxed and discrete QAP. Given an initial continuous or binary candidate solution $y(0)$, it improves iteratively the corresponding quadratic objective function ($S(y) = y^T K y$). At each iteration, a LSAP problem needs to be solved.

3.1.1.3.2.7 Path following methods Graduated NonConvexity and Concativity Procedure (GNCCP) [Liu and Qiao, 2014, Zaslavskiy et al., 2009, Bougleux et al., 2017a] is a path following algorithm which aims at approximating the solution of a QAP by considering a convex-concave relaxation through the modified quadratic function: GNCCP–Graduated NonConvexity

and Concavity Procedure.

$$S_\xi(y) = (1 - |\xi|)S(y) + \xi(y^T y)$$

where $\xi \in [-1, 1]$. When $\xi = 1$, $S_\xi(y) = y^T y$ is fully convex, and when $\xi = -1$, $S_\xi(y) = -y^T y$ is concave. GNCCP algorithm starts with $\xi = -1$ and it leads to the maximization of a concave problem. Any initial solution can be chosen to solve the concave problem and it has no influence on the result. So, unlike IPFP algorithm, no initial matching is required. Then GNCCP algorithm smoothly interpolates concave and convex relaxations by iteratively increasing ξ from -1 to 1 with step size α (equal to 0.1). For each iteration corresponding to a ξ, the maximization of S_ξ is achieved by IPFP.

3.1.1.3.2.8 Spectral graph matching Spectral methods consist of studying the similarities between the spectra of the adjacency or Laplacian matrices of the graphs and using them for matching. The matrix $K \in \mathbb{R}^{|V_1|.|V_2| \times |V_1|.|V_2|}$ (see Model SGMIQP) is used for this purpose. This matrix has to be non-negative, symmetric. K can be seen as a weighted adjacency matrix of the graph $G_X = G_1 \times G_2$ where G_x is obtained by direct graph product. Given two attributed graphs $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (V_2, E_2, \mu_2, \zeta_2)$, G_X is the complete non-directed graph that associates a vertex to each couple $(i, k) \in V_1 \times V_2$. The problem is then recast to a **node clustering problem** that is solved by spectral method using the principal eigenvector of K and imposing the mapping constraints (one-to-one mapping) [Leordeanu and Hebert, 2005, Cour et al., 2007]. Similarly, the Laplacian matrix ∇_X of G_X can be computed. The second-smallest eigenvalue of Ω_X is called Fiedler eigenvalue. The eigenvector associated with Fiedler eigenvalue has been named the Fiedler vector. The Fiedler vector can be used to partition a graph. The negative values are associated with the poorly connected cluster while the positive values are associated with the connected cluster. The signs of the values in the Fiedler vector can therefore be used to partition this graph into two clusters. Eigenvectors are independent to the permutation of vertices in the matrices ∇_X and K .

3.1.1.3.2.9 MAP-inference-based methods MAP-inference problem (Problem 8) on a discrete pairwise graphical model, also called Conditional Random Field (CRF) in the literature, is related to error-tolerant graph matching (see Problem 9). It differs in an additional uniqueness constraint: Each label can be taken only once. Also, the graph matching problem, after possibly introducing many additional variables, can be stated as a MAP-inference problem in a standard pairwise CRF. The uniqueness constraint prevents naive application of efficient solvers for MAP-inference to this problem. For this reason, many dedicated graph matching solvers were developed. On the other hand, efficient dual block-coordinate ascent (also known as message passing) algorithms like TRW-S [Kolmogorov, 2006] count among the most efficient solvers for MAP-inference in conditional random fields. The key idea is to use techniques from the MAP-inference community to gain computational efficiency. On the high level, the idea is to decompose the original problem into several "easier" subproblems, for which an efficient global minimum (or a good lower bound) can be computed. Combining the lower bounds for individual subproblems will then provide a lower bound for the original problem. The decomposition and the corresponding lower bound will depend on a parameter vector. The goal is to find a vector that maximizes the bound. This approach is well-known in combinatorial optimization; sometimes it is referred to as "dual decomposition". The dual decomposition solver [Torresani et al., 2013] represents the problem as a combination of MAP-inference for binary CRFs (labels are 0 or 1), or a combination of linear assignment problems or a combination of small-sized QAPs. Lagrange multipliers connecting these subproblems are updated with the sub-gradient method [Kappes et al., 2012, Zhang et al., 2016a, Wright, 2015].

3.1.1.3.2.10 Factorized graph matching Factorized graph matching is a framework for interpreting and optimizing graph matching problems. The affinity matrix K (see Model SGMIQP)

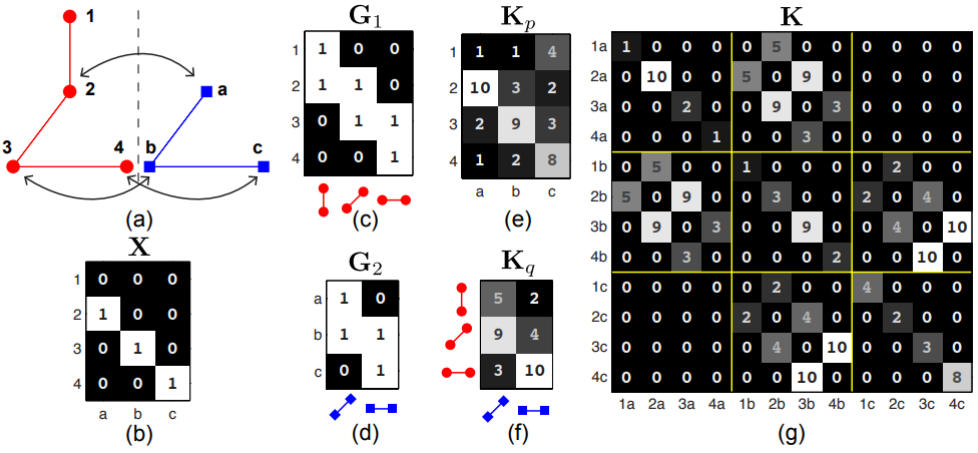


Figure 3.4: Example of graph matching and related matrices [Zhou and la Torre, 2016]. (a) Two synthetic graphs. (b) The correspondence matrix X. (c) The first graph's incidence matrix G_1. (d) The second graph's incidence matrix G_2. (e) The node affinity matrix K_p. (f) The edge affinity matrix K_q. (g) The global affinity matrix K.

can be factorized as a Kronecker[1] product of smaller matrices. There is a main benefits of using this factorization in graph matching: There is no need to compute the costly (in space and time) pair-wise affinity matrix. K can be factorized because

- it is organized in $|\mathcal{V}_2| \times |\mathcal{V}_2|$ blocs of size $|\mathcal{V}_1| \times |\mathcal{V}_1|$.
- Many K_{ij} contain only zero-value elements and their positions are indexed by $\mathcal{V}_2$.

The full details of the factorization can be found in [Zhou and la Torre, 2016]. We propose to explain the key elements of the decomposition as follows :

$$K \propto Structure \times Affinity$$

This principles is presented in Figure 3.4. Observe that this factorization decouples the graph structure from the pairwise similarity. The *Structure* is expressed by products of incident matrices of G_1 and G_2 while $Affinity$ hold two sub-matrices to denote vertex-to-vertex similarities and edge-to-edge similarities independently.

Such a decomposition avoids the computation of the cumbersome affinity matrix (K) and hence potentially allows for a more efficient implementation, especially for large graphs. The factorization leads to a new heuristic of the subgraph matching problem.

3.1.1.3.2.11 Evolutionary algorithms (non-deterministic) Evolutionary Algorithms (EAs) are nature inspired heuristics, which are widely used to tackle many $\mathcal{NP}$-hard problems. The key idea behind EAs is mimicking the rule "survival-of-the-fittest" on a population of different individuals. Two major EAs, genetic algorithms and ant colonies, have been applied to solve the graph matching problems. In genetic algorithms [Riesen, 2015, Cross et al., 1997, Bengoetxea et al., 2002], an individual is a set of integers pe as in Problem QAPGM. pe represents a solution. The fitness function measures the suitability of an individual. To do so, the objective function

[1] If A is an m × n matrix and B is a p × q matrix, then the Kronecker product A ⊗ B is the mp × nq block matrix

of the graph matching problem is a usual choice. If pe is an unfeasible solution then a very high cost is associated to pe. The group of individuals (also known as population) evolves towards more promising areas of the search space while the algorithm carries on with the next generation. New individuals result from inheriting parts of solutions from its parents. Iteratively, the new population of individuals is generated by using crossover or mutation operators. For instance, a mutation is applied to a single individual, a swap between two elements in pe is performed. It can be interpreted as a change of the two matchings $(i \to k)$ and $(j \to l)$ in $(i \to l)$ and $(j \to k)$. These principles have been applied to solve the GED problem in [Ibragimov et al., 2013, Riesen, 2015].

In [Sammoud et al., 2005], an Ant Colony Optimization (ACO) algorithm for solving graph matching problems is proposed. The main idea is to model the problem as the search for a minimum cost path in a graph —called construction graph— and to use artificial ants to search for good paths. The construction graph G_X is the complete non-directed graph that associates a vertex to each couple $(i, k) \in V_1 \times V_2$. The behavior of artificial ants is inspired from real ants. They lay pheromone trails on graph components and they choose their path with respect to probabilities that depend on pheromone trails that have been previously laid. These pheromone trails progressively decrease by evaporation. Intuitively, this indirect communication aims at giving information about the quality of path components in order to attract ants, in the following iterations, towards the corresponding areas of the search space. The amount of pheromone on an edge $((i, k), (j, l))$ represents the desirability of matching together $i \to k$ and $j \to l$.

3.1.1.3.2.12 Probabilistic framework (non-deterministic) In [Myers et al., 2000], a Bayesian Graph Edit Distance is proposed. Myers et al shows how the Levenshtein distance can be used to model the probability distribution for structural errors in graph matching problems. Let f be a matching function $f : \mathcal{V}_1 \to \mathcal{V}_2$. The posterior probability of f given G_1 and G_2 can be written as follows:

$$pr(f|G_1, G_2) = \frac{Pr(G_1, G_2|f)p(f)}{Pr(G_1, G_2)}$$

The local optimum of the *a posteriori* probability may be located by applying the following iterative decision rule to update the matching configuration:

$$f(u) = \arg\max_{v \in \mathcal{V}_2} \frac{Pr(u, v|\mu_1(u), \mu_2(v))}{Pr(u, v)} Pr(f) \quad \forall u \in \mathcal{V}_1$$

In [Zass and Shashua, 2008], a probabilistic model is presented for soft graph matching between complex feature sets. The key idea is transform the matrix K into a probability matrix for any possible match between the edges and vertices between G_1 and G_2. K is modified to become a doubly stochastic matrix. To turn compatibility measures into probabilities, the Sinkhorn algorithm [Sinkhorn and Knopp, 1967] for finding the nearest doubly stochastic matrix is performed.

In [Cho et al., 2010a], the graph matching solution is obtained by simulating random walks in $G_X = G_1 \times G_2$ that is the direct graph product between G_1 and G_2. K can be seen as the weighted adjacency matrix of G_X taking into account affinities of the graph component correspondences. K is then row-normalized such that each row sums to one ($\tilde{K}$). Therefore, $\tilde{K_{ij}}^2$ provides, from node i to node j, the probability of walks of length 2 based on affinities. By extending this principle to the n-th power and searching for the most probable path, the algorithm achieves noise-robust graph matching.

The last type of non-deterministic methods is called Estimation of distribution algorithms (EDAs). EDAS combine two technical disciplines of soft computing methodologies: probabilistic reasoning and evolutionary computing. In brief, EDA are population-based search algorithms based on probabilistic modelling of promising solutions. In EDA the new population of individuals is not generated by using crossover nor mutation operators. Instead, the new individuals are

ETSGM	Methods		
Models		Exact	Heuristique
	ILP	#papers : 0	#papers : 0
	IQP	#papers : 0	#papers : 7 IPFP [Leordeanu et al., 2009]
ECGM	**Methods**		
Models		Exact	Heuristique
	ILP	#papers : 0	#papers : 1 ILP [Justice and Hero, 2006]
	IQP	#papers : 0	#papers : 3 mIPFP [Bougleux et al., 2017b]

Figure 3.5: State of the art on graph matching. #papers is the number of papers according to the criteria: model type and heuristic or exact methods. A paper is cited for each category to give an example.

sampled starting from a probability distribution estimated from the database containing only selected individuals from the previous generation. In [Bengoetxea et al., 2002, 2001], EDAs are used to solve matching problems.

3.1.1.4 Summary

To conclude, the methods are tabulated in Table 3.1 according the following criteria:

- The graph matching problem to be addressed.
- The related problem (QAP or MAP-inference problems).
- The mathematical model used to represent the problem.
- Is the method an heuristic or an exact method ?
- Is the method deterministic or not ?
- The method family (tree-based, spectral, evolutionary, ...).

Figures 3.5 and 3.6 give a synthetic view of the literature. Figure 3.5 displays the number of research papers according to the model they use and the method family. Figure 3.6 shows the number of research papers by the solving method types. From these figures two facts can be stated. Exact methods are rarely study (only 1 paper) and few works have paid attention to ILP models.

From Table 3.1, we can observe that people working on Problem 3(ETSGM) have concentrated their efforts on the QAP and MAP-inference solvers (Frank-Wolfe like methodology, dual decomposition methods, ...). Research community working on Problem 4(ECGM) have favored LSAP-based and tree-based methods.

3.1.1.5 Discussions on the literature of graph matching

The discussion is organized in three main steps. First, a discussion is led about the Model GMIQP and Model SGMIQP. Theses two models have split the researchers into two communities working on different methods. It is interesting to study the relation between the two models. Second, performance evaluation metrics are put forward along with the data sets. Third, considerations about efficiency and effectiveness of the methods are highlighted.

Methods	GM Problem	Higher Problem	Model	Exact or heuristic	Determinist	Method Family
IPFP [Leordeanu et al., 2009]	ETSGM	QAP	IQP	heuristic	yes	Frank-Wolfe like
Factorized Graph Matching [Zhou and la Torre, 2016]	ETSGM	QAP	IQP	heuristic	yes	Iterative + Frank-Wolfe like
PATH [Zaslavskiy et al., 2009]	ETSGM	QAP	IQP	heuristic	yes	Iterative + Frank-Wolfe like
GNCCP [Liu and Qiao, 2014]	ETSGM	QAP	IQP	heuristic	yes	Iterative + Frank-Wolfe like
SM[Leordeanu and Hebert, 2005]	ETSGM	QAP	IQP	heuristic	yes	Spectral
SMAC [Cour et al., 2007]	ETSGM	QAP	IQP	heuristic	yes	Spectral
Probabilistic [Zass and Shashua, 2008]	ETSGM	QAP	IQP	heuristic	no	Sinkhorn algorithm
DD [Torresani et al., 2013]	ETSGM	MAP-inference	IQP	heuristic	yes	Dual decomposition
Hungarian-BP [Zhang et al., 2016a]	ETSGM	MAP-inference	IQP	heuristic	yes	Dual decomposition
DDLagrange [Kappes et al., 2012]	ETSGM	MAP-inference	IQP	heuristic	yes	Dual decomposition
A* [Riesen et al., 2007]	ECGM			exact	yes	Tree-based
BP [Riesen and Bunke, 2009]	ECGM			heuristic	yes	Hunagrian like
FBP [Serratosa, 2015]	ECGM			heuristic	yes	Hunagrian like
LSAPE [Bougleux et al., 2017b]	ECGM			heuristic	yes	Hunagrian like
GeneticSearch [Riesen, 2015]	ECGM			heuristic	no	Genetic algorithm
GEDEVO [Ibragimov et al., 2013]	ECGM			heuristic	no	Genetic algorithm
LocalSearch [Riesen, 2015]	ECGM			heuristic	yes	Beam Local Search
BS [Neuhaus and Bunke., 2007]	ECGM			heuristic	yes	Beam Search
Bayesian GED [Myers et al., 2000]	ECGM			heuristic	no	Expectation Maximization
mIPFP [Bougleux et al., 2017a]	ECGM	QAP	IQP	heuristic	yes	Frank-Wolfe like
mGNCCP [Bougleux et al., 2017a]	ECGM	QAP	IQP	heuristic	yes	Iterative + Frank-Wolfe like
ILP [Justice and Hero, 2006]	ECGM	QAP	ILP	exact and heuristic	yes	black Box solver
AntAlgo [Sammoud et al., 2005]	Multi-valent			heuristic	no	Ant Colonny

Table 3.1: Graph matching literature

Methods	ECGM	ETSGM
Constrained quadratic programming	#papers : 1 mIPFP [Bougleux et al., 2017b]	#papers : 4 – IPFP [Leordeanu et al., 2009]
Spectral	#papers : 0	#papers : 2 SM[Leordeanu and Hebert, 2005]
Branch and bound	#papers : 2 A* [Riesen et al., 2007]	#papers : 0
Bio-inspired	#papers : 2 GEDEVO [Ibragimov et al., 2013]	#papers : 2 AG [Cross et al., 1997]
Hungarian method	#papers : 3 SFBP [Serratosa, 2015]	#papers : 0
Probabilistic	#papers : 1 BayesianGED [Myers et al., 2000]	#papers : 1 [Zass and Shashua, 2008]

Figure 3.6: State of the art on graph matching.#papers is the number of papers according to the type methods. A paper is cited for each category to give an example.

3.1.1.5.1 Model GMIQP*vs* Model SGMIQP Here we would like to discuss the relation between Model GMIQP and Model SGMIQP. Model GMIQP models Problem ECGM while Model SGMIQP models Problem ETSGM. First to make a fair comparison, let us assume that the cost function ($c(\cdot)$) is related to the affinity function ($s(\cdot)$) by a large constant cst as follows: $s = cst - c$. Without loss of generality, let us write down the flowing hypothesis : $c(.) \geq 0$ and $s(.) \geq 0$. The costs to delete or to insert vertices or edges are explicitly introduced in Model GMIQP. In Model SGMIQP, deletion costs are implicitly set to 0.

Proposition 1. *Solving Model GMIQP and Model SGMIQP is equivalent in terms of solution for any arbitrary cost function.*

Sketch of proof. If we can prove that Proposition 1 is wrong for one example then we can admit that Proposition 1 is wrong in the general case. Let G_1 be a graph with a single vertex (i) and no edges. Let G_2 be a graph with a two vertices (k and l) and no edges as depicted in Figure 3.7. Let us define the costs as follows : $c(i \to k) = 0$, $c(i \to l) = 10$, $c(i \to \epsilon) = 10$, $c(\epsilon \to k) = 10$, $c(\epsilon \to l) = 30$. Since inserting k is cheaper than inserting l, the optimal solution of Model GMIQP is $i \to l$ and $\epsilon \to k$ with a total cost of 20. Let $cst = 100$ then the compatibilities are as follows: $s(i \to \epsilon) = 100 - 10$, $s(\epsilon \to k) = 100 - 10$, $s(\epsilon \to l) = 100 - 30$, $s(i \to k) = 100 - 0$ and $s(i \to l) = 100 - 10$. The optimal solution of Model SGMIQP is $i \to k$. The two solutions are different so Proposition 1 is wrong. In this case, deletion costs are discriminative and so they are important to perform the matching. □

Proposition 2. *Solving Model GMIQP and Model SGMIQP is equivalent in terms of solution if deletion and insertion costs are identical for any vertex or edge and higher than any substitution cost.*

Sketch of proof. This is the specific case of the MCS problem (Problem 2) as stated in [Bunke, 1997, 1999, Brun et al., 2012]. Both models can express the MCS problem. Since deletion and insertion costs are identical and higher than any substitution, an explicit or implicit definition of deletion and insertion costs are equivalent. Proposition 2 is right. An example is presented in Figure 3.8. □

3.1.1.5.2 Performance evaluation metric Performance evaluation is a major subject. In most of the cases, three performance evaluation metrics are used. The first metric is the called "Deviation" or "Objective Ratio". It is the relative error between the objective function values

Operation	Cost	Similarity
i → k	0	100
i → l	10	90
$\varepsilon_l \rightarrow l$	30	NA
$\varepsilon_k \rightarrow k$	10	NA
$i \rightarrow \varepsilon_i$	10	NA

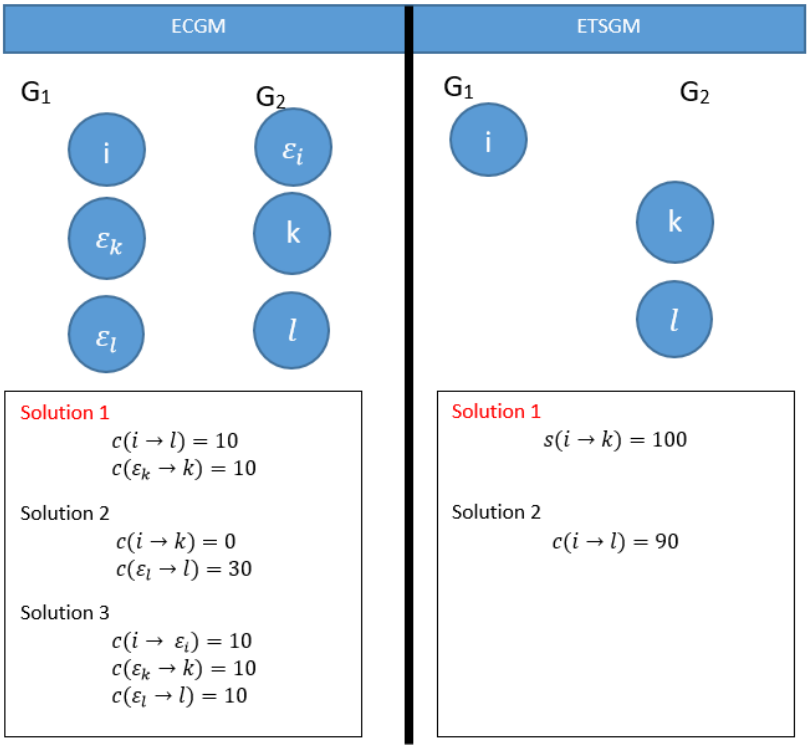


Figure 3.7: A comparison of the sugbraph matching and error-correcting graph matching problems. The solutions 1 are the best in the two cases.

Operation	Cost	Similarity
i → k	0	100
i → l	10	90
$\varepsilon_l \rightarrow l$	30	NA
$\varepsilon_k \rightarrow k$	30	NA
$i \rightarrow \varepsilon_i$	30	NA

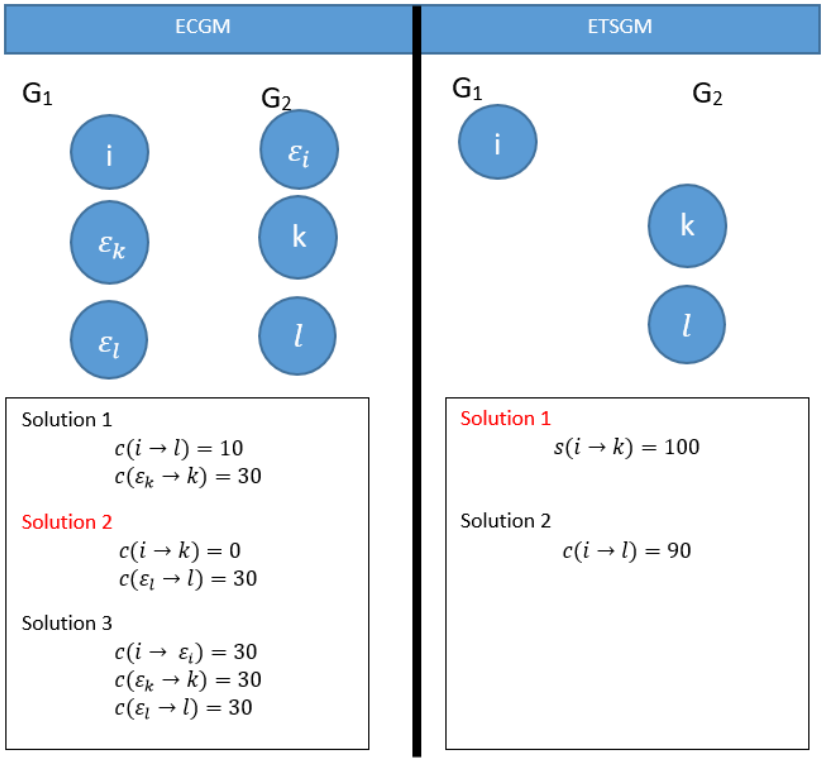


Figure 3.8: A comparison of the sugbraph matching and error-correcting graph matching problems when deletion and insertion costs are identical for any vertex or edge and higher than any substitution cost.

of a given method and a reference method on a single instance (a pair of graphs). Let $\mathcal{D}$ be a graph data set that consists of M graphs, $\mathcal{D} = \{G_1, G_2, ..., G_M\}$. Let $\mathcal{P}$ be the set of all of the methods. Given a method $pm \in \mathcal{P}$, the square distance matrix $Q^{pm} \in \mathcal{M}_{M\times M}$, which holds every pairwise comparison $Q^{pm}_{i,j} = d_{pm}(G_i, G_j)$, where the distance $d_{pm}(G_i, G_j)$ is the value returned by the method pm on the graph pair (G_i, G_j).

$$deviation(i,j)^{pm} = \frac{|Q^{pm}_{i,j} - R_{i,j}|}{R_{i,j}}, \forall i,j \in [1,\cdots,M], \forall pm \in \mathcal{P} \tag{3.7}$$

where $R_{i,j}$ is defined in Equation 3.8.

$$R_{i,j} = \min_{pm\in\mathcal{P}} \{Q^{pm}_{i,j}\},\ \forall i,j \in [1,\cdots,M] \tag{3.8}$$

$R_{i,j}$ is either the best upper bound or the optimal solution (if available). For a given method, the deviation can express the error made by a method in terms of the percentage of the best method. The second metric is called "accuracy" or "Dissimilarity Matching" in the literature. It denotes the dissimilarity between a computer-generated matching ($y_{(computer)}$) and a Human-made matching ($y_{(human)}$). A matching is represented by a vector of binary values (y) so the dissimilarity matching can be computed as the Hamming distance between $y_{(computer)}$ and $y_{(human)}$.
The last metric is the speed of a given method.

When comparing a computer generated matching and the human ground-truth, there could be two reasons to explain missmatches. The first reason is that graph matching solver is not an exact method and thus the computed matching is not the optimal one. The suboptimal solution is far from the expected matching. A second reason is that the graph matching solver has computed a very good solution (the best minimum) but the defined cost functions are not in adequacy with the Human need or goal. Consequently, dissimilarity Matching is not a good criterion to assess graph matching solvers. Bridging the gap between Human ground-truth and computer-generated matchings is more linked to the machine learning community. However, fast and effective solvers are a key element in this objective.

A specific metric appears when solving Problem ECGM. Solving Problem ECGM leads to a dissimilarity measure between graphs. This dissimilarity measure can be involved in a classification step. The classification rate is then used as a metric to gauge the solver effectiveness. The assumption is that better the solver higher the classification rate. This assumption is wrong and an heuristic can give a higher classification rate than an exact method. It is dependent on the data distribution and the classifier. It will be discussed in the graph classification section (Section 3.2.

3.1.1.5.3 Data sets Data sets to evaluate graph matching methods can be analyzed through three different angles. First, the question of the ground-truth matching availability. Three cases can appear:

1. The ground-truth matching is available.
2. The ground-truth matching is not available.
3. The ground-truth matching is indirectly available. It is available through some properties of the final application. The ground-truth is usually computed by an application-dependent method.

Second, data sets can also be split according to the cost function view point :

1. The cost function is provided and so the comparison is reproducible.

2. The cost function is not defined and it should be found to fit the application need.

Finally, graph matching methods are impacted by some intrinsic characteristics of the datasets :

1. The number of vertices.
2. The connectivity or the density of the graphs: A dense graph is a graph in which the number of edges is close to the maximal number of edges. At the opposite, a graph with only a few edges is a sparse graph. The distinction between sparse and dense graphs is rather vague, and depends on the context. For undirected simple graphs, the graph density is defined as: $Density = \frac{2|E|}{|V|\,(|V|-1)}$
3. The type of cost functions: Cost functions can be very different. Especially, their outputs can be binary (0 or 1) or continuous and everything in between. In addition, the discriminative power or the distribution of the costs according to the graph component pairs is crucial for graph matching solvers. Roughly speaking and to give an example, a discriminative cost function is a function answering 0 for a given pair of vertices $(i \rightarrow k)$ and high values for other vertex matches $(i \rightarrow l)$. A cost function that discriminates between vertices can be helpful. At the opposite, a cost function making each vertex comparison identical is somehow not informative for the solver and so it makes the graph matching more challenging. A binary cost function re-casts the problem to an exact matching problem.

Below, we propose a list of the main data sets used for error-tolerant graph matching. These datasets have been chosen by carefully reviewing all of the publicly available datasets that have been used in the reference works mentioned in the state of the art section.

1. CMU House/Hotel: house [cmu] and hotel [cmu] with costs as in [Torresani et al., 2013]. The task is to find a matching between two images. This dataset consists of 111 frames of a house, each of which has been manually labeled with 30 landmarks. The Delaunay triangulation is used to connect the landmarks. Each frame represents the same object but with different rotation angles. Frame number 0 and frame number 10 represent the same object but with a rotation of 10 degrees. Intuitively, the gap between rotation angles controls the matching difficulty.
2. VOC car and motorbike: in [Leordeanu et al., 2012], this data sets contain pairs of cars and motorbikes with keypoints to be matched. Delaunay triangulation is used to connect the keypoints inside an image. The images are taken from the VOC PASCAL 2007 challenge. Costs are computed from features [Leordeanu et al., 2012]. Graphs contain outliers (points of the background or from other objects). Intuitively, the number of outliers controls the matching difficulty.
3. Graph flow: the graph flow dataset [gra, 2015] comes from a tracking problem with large displacements [Abu Alhaija et al., 2015]. Keypoints in frames of RGB-D images obtained by a Kinect camera are matched. The depth information provided by the Kinect camera is taken into account when computing the cost functions.
4. MUTA [Riesen and Bunke, 2008] ata set comes from the IAM Graph Database Repository [Riesen and Bunke, 2008]. Alkane, Acyclic, PAH, MAO are from the GREYC database repository [gre]. These graphs are mainly purely structural datasets representing chemical molecules. Vertices represent atoms and edges are valence bounds.
5. GREC data set comes from the IAM Graph Database Repository [Riesen and Bunke, 2008]. The GREC data set consists of graphs representing symbols from architectural and electronic drawings. The images occur at five different distortion levels.

Dataset	# graphs	$\overline{\|V\|}$	$\overline{\|E\|}$	max |V|	Density	GT	Cost	Application	Cost type
MUTA	4337	30.3	30.7	417	2.02	no	yes	Classification	Binary
Acyclic	185	8.1	7.1	11	1.7	no	yes	Classification	Binary
Alkane	150	8.8	7.8	10	1.78	no	yes	Classification	Binary
MAO	68	18.3	19.6	27	2.13	no	yes	Classification	Binary
PAH	94	20.7	24.4	28	2.3	no	yes	Classification	Binary
GREC	1100	11.5	11.9	24	2.0	no	yes	Classification	Continuous
LETTER	3x750	4.7	3.9	9	0.8	no	yes	Classification	Continuous
CMU House	111	30	79.1	30	5.27	yes	yes	Classification	Continuous
CMU Hotel	105	30	79.1	30	5.27	yes	yes	Classification	Continuous
VOC CAR	30	34	85	49	5.77	yes	yes	Matching	Continuous
VOC MOTO	20	33.5	76	52	5.45	yes	yes	Matching	Continuous
Graph flow	6	88		126	sparse	indirectly	no	Matching	-
CUB	11 000	200		256	dense	indirectly	no	Matching	-

Table 3.2: Dataset description for graph matching assessments.

6. LETTER [Riesen and Bunke, 2008]: LETTER is broken down into three parts (LOW, MED, HIGH) which corresponds to distortion levels. Assessing methods according to the noise level is an interesting viewpoint when dealing with pattern recognition problems. The LETTER dataset is useful because it holds graphs of rather small size (i.e maximum of 9 nodes). This property is interesting to compute optimal solutions.
7. CUB [Zanfir and Sminchisescu, 2018]: This dataset contains 11,788 images of 200 bird categories, with bounding box object localization and 15 annotated key points per image.
8. Most of the databases come can be found on the IAPR TC-15 website[2]

The main characteristics of the data sets are tabulated in 3.2.

3.1.1.5.4 Graph matching library Here are some of the graph matching libraries :

- Path Following: http://projects.cbio.mines-paristech.fr/graphm/
- Factorized graph matching: http://www.f-zhou.com/gm_code.html
- IPFP, SMAC, SM: https://sites.google.com/site/graphmatchingmethods/
- Message passing methods: https://github.com/pawelswoboda/LP_MP-QAP and http://paulswoboda.net/publications
- LSAP: https://bougleux.users.greyc.fr/lsape/
- RRWM: https://cv.snu.ac.kr/research/~RRWM/
- QAPLib: http://anjos.mgi.polymtl.ca/qaplib/
- GED Evolutionary: http://gedevo.mpi-inf.mpg.de/
- Feature Correspondence via Graph Matching: Models and Global Optimization: http://pub.ist.ac.at/~vnk/software/GraphMatching-v1.02.src.zip
- MAP CRF solvers: http://hciweb2.iwr.uni-heidelberg.de/opengm/index.php

The large volume of code available is an index to measure the maturity of the community. The problems, the input and output are well defined and it makes comparisons between methods easier.

[2] http://www.greyc.ensicaen.fr/iapr-tc15/index.php

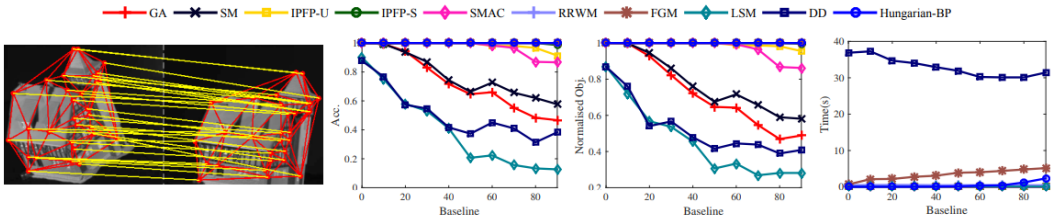


Figure 3.9: From [Zhang et al., 2016a]. The x-axis is the rotation angle gap between frames. The "Acc" axis represents the similarities between matchings. The normalized Obj axis represents the normalized objective function values.

3.1.1.5.5 Discussion on the running times In this paragraph, our ambition is not to benchmark all the methods but rather to provide intuition about the speed of the methods.

First let us take a look to methods solving Problem ETSGM. PATH [Zaslavskiy et al., 2009], IPFP [Leordeanu et al., 2009], GNCCP [Liu and Qiao, 2014] call the Hungarian method at each iteration. Factorized Graph Matching (FGM) [Zhou and la Torre, 2016] relies on the expensive of the Frank-Wolfe algorithm. All these methods have a worst case of $O(n^3)$ per iteration time complexity. The Dual Decomposition (DD) [Torresani et al., 2013] algorithm is more computational expensive. In the papers of recent graph matching algorithms [Zaslavskiy et al., 2009, Zhou and la Torre, 2016, Leordeanu et al., 2009, Liu and Qiao, 2014, Cour et al., 2007, Leordeanu and Hebert, 2005, Torresani et al., 2013, Kolmogorov, 2006], the experiments were done on graphs with a number of nodes from 20 to 200.

Second let us take a look to methods solving Problem ECGM. Many fast heuristics have been designed. In [Riesen and Bunke, 2009], the memory requirements and execution times of this method are respectively proportional to $(n_1 + n_2)^2$ and $(n_1 + n_2)^3$ where n_1 and n_2 are the order of the graphs. In [Bougleux et al., 2017b], for the same results, the algorithm requires $O(n_1 n_2)$ memory space and $O(\min(n_1, n_2)^2 \max(n_1, n_2))$ execution times. In [Serratosa, 2015], a reduction of the the execution times $O(\min(n_1, n_2)^3)$ is achieved but at the price of an approximated solution of the LSAP problem. Tree-based methods such as BeamSearch can be very fast if the beam size is small. To the extreme, if the beam size is equal to one then the method is called a greedy search and the exploration is limited to a single branch of tree. Other methods tend to be more time consuming but it is not possible to conclude about a clear ranking.

3.1.1.5.6 Effectiveness analysis This paragraph is not about an exhaustive evaluation of the methods but we compare the most frequent methods of the literature on two important and challenging data sets. First let us take a look to methods solving Problem ETSGM. Results are shown in Figure 3.9 about the house CMU data set. The task is to find a matching between two images. This dataset consists of 111 frames of a house, each of which has been manually labeled with 30 landmarks. The Delaunay triangulation is used to connect the landmarks. Each frame represents the same object but with different rotation angles. Frame number 0 and frame number 10 represent the same object but with a rotation of 10 degrees. Intuitively, the gap between rotation angles controls the matching difficulty. In this experiment, the node-affinity $K_{ik,ik}$ was set to zero and the edge-affinity $K_{ik,jl}$ was set to $K_{ik,jl} = \exp(-\frac{(\zeta(ij)-\zeta(kl))^2}{2500})$ where $\zeta(ij)$ is the distance between two keypoints in the image. In Figure 3.9, note that as the separation between frames increases, the accuracy of several algorithms drops precipitously. The four methods: IPFP-S, RRWM, FGM, and Hungarian-BP exactly identify the correct match in all scenarios.

Results are shown in Figure 3.10 about the VOC data sets. Each node feature $\mu(i)$ is an orientation angle. Each edge was represented by a couple of values, $\zeta(ij) = [d, \theta]$, where d is the

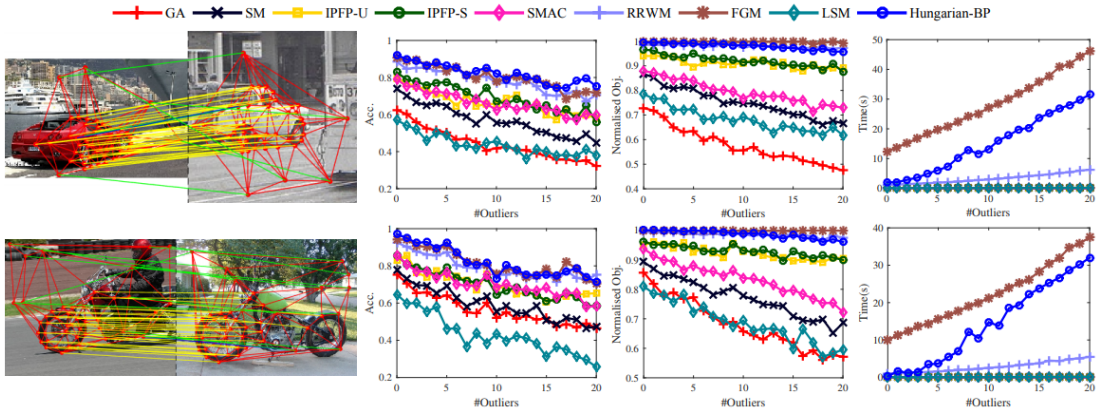


Figure 3.10: From [Zhang et al., 2016a]. Matching results on VOC images. Typical matching result are shown on the left. Yellow lines indicate correct matches, blue lines indicates incorrect matches, and green lines indicate matches between outliers. The results of DD are not shown due to the prohibitive execution time

	BS-100	LSAPE	mIPFP
Deviation (%)	15	80.6	0
Time (s)	0.779	0.144	0.191

Table 3.3: Results on PAH dataset of error-correcting graph matching methods with a time limit of 300s.

pairwise distance between the connected nodes and θ is the absolute angle between the edge. Thus, for each pair of images, we computed the node affinity as $K_{ik,ik} = \exp(-||\mu(i) - \mu(k)||_1)$ and the edge affinity as $K_{ik,jl} = \exp(-\frac{1}{2}||d_{ij} - d_{kl}||_1 - \frac{1}{2}||\theta_{ij} - \theta_{kl}||_1)$. VOC datasets contains vertex and edge outliers. Without outliers, Hungarian-BP always achieves the highest accuracy. It also achieves the best objective in the "Motorbikes" dataset, and the second best in the "Car" dataset. The speed of the Hungarian-BP is also quite competitive. In the Motorbike dataset, DD achieves the second best accuracy, but its speed is hundreds of times slower than that of Hungarian-BP method. In the Car dataset, FGM achieves the second best accuracy, but its speed is 10 times slower than that of Hungarian-BP. When outliers exist, the running time of Hungarian-BP algorithm increases with the number of outliers. However it is sill faster than the FGM method.

Second let us take a look to methods solving Problem ECGM. Results on PAH and CMU House data sets are reported on Table 3.3 and Table 3.4, respectively. Costs associated with these experiments are reported in Table 3.5. According to these results, mIPFP seem to be the fastest and the most accurate. mIPFP dominates other methods on the two data sets.

3.1.1.5.7 Reasoning about effectiveness and speed-up

	BS-5	SBPBeam-5	mIPFP	mGNCCP
Deviation (%)	103.6	5.24	0	20
Time (s)	0.14	8.50	0.18	9.61

Table 3.4: Results on CMU House dataset of error-correcting graph matching methods with a time limit of 10s.

Operation cots	PAH	CMU-House
$c(i \to k)$	1	0
$c(i \to \epsilon)$	3	1000
$c(\epsilon \to k)$	3	1000
$c(ij \to kl)$	1	$\|\|\zeta(ij) - \zeta(ij)\|\|_1$
$c(ij \to \epsilon)$	3	$\|\|\zeta(ij)\|\|_1$
$c(\epsilon \to kl)$	3	$\|\|\zeta(ij)\|\|_1$

Table 3.5: Cost functions for PAH and CMU House data sets.

As it is usual for $\mathcal{NP}$-hard problems, no single method can effectively address all QAP instances. Different applications require different methods and we concentrate here on problem instances specific for computer vision and pattern recognition. Traditionally, within this community predominantly heuristics are used, since demand for low computational time usually dominates the need to obtain optimality guarantees.

3.1.2 Open problems

As previously stated, the computer vision and pattern recognition are mainly focused on heuristic methods. Consequently, exact methods are not well studied in the literature. There is only one ILP [Justice and Hero, 2006] to solve the error-correcting graph matching problem but it does not solve the general case because edge attributes are not taken into account. However, exact methods are important because they can help to evaluate heuristics methods in terms of effectiveness. In addition, exact methods can be used to solve efficiently "not large" instances. To finish on this aspect, heuristics can be derived from exact methods. For all theses reasons, we think that it is important to study exact methods. Graph matching benchmarks are well-established in the community, especially, VOC and CMU databases. However, these data sets do not reflect all the parameters that impact graph matching methods such as:

1. The number of vertices.
2. The density of the graphs.
3. The type of cost functions.

There is a room to enrich graph matching benchmarks to better fit with real applications as well as better characterize the behaviour of graph matching solvers. Finally, researchers working on error-correcting graph matching and error-tolerant subgraph matching problems are not the same. They form two distinct communities with their own benchmarks and methods. It is crucial to bridge the gap between the Problem ETSGM and Problem ECGM. A first step forward has been done by [Bougleux et al., 2017a] by modelling Problem ECGM as a QAP and using solvers from the subgraph matching community. In this direction, more investigations could be led to compare ETSGM and ECGM problems. The goal would be to unify methods and benchmarks. To sum up, here are the fours deadlocks to be opened:

1. There is a need to study exact methods.
2. There is a need to work on the performance evaluation of graph matching methods.
   (a) Performance evaluation of heuristics with respect to optimal solutions (thanks to deadlock 1).
   (b) To enrich graph matching benchmarks and to better characterize their behaviors.

3. Like any $\mathcal{NP}$-hard problem there is always a trade-off to be found between speed and effectiveness. There is a need to study heuristics according to speed and effectiveness criteria.

4. There is a need to draw links between Problem ETSGM and Problem ECGM to gather people from both communities.

3.1.3 Contribution

The first deadlock to be released is about exact methods. It is addressed through the design of new models and a branch and bound methods. The second deadlock is about the performance evaluation. A benchmark was proposed and latter used in an international contest. Accurate and fast heuristics is a *Graal*, we proposed to develop accurate and flexible heuristics to cope with a wide range of applications. Finally, we propose a theoretical study to relate Model SGMIQP and Model GMIQP.

3.1.3.1 Deadlock 1: models and algorithms for exact methods

3.1.3.1.1 Motivation Exact methods to solve Problem ECGM are not well studied. As an illustration, computing the optimal solution of Problem ECGM using A^* is only feasible for graphs of a rather small size (typically 10 vertices) because of the its memory consumption. Another exact method exists named JH [Justice and Hero, 2006], it is based on ILP but does not solve the general problem. We tackle the question of designing exact methods for Problem ECGM by adopting two strategies: the design of new ILP models and the design of memory efficient algorithms.

3.1.3.1.2 Exact methods: mathematical models A BLP is a restriction of integer linear programming (ILP) where the variables are binary. Hence, its general form is :

$$\min_x \ c^T x \tag{3.9a}$$

$$\text{subject to } Ax \leq b \tag{3.9b}$$

$$x \in \{0,1\}^n \tag{3.9c}$$

where $c \in \mathbb{R}^n, A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^m$ are data of the problem. A feasible solution is a vector x of n binary variables (3.9c) which respects linear inequality constraints (3.9b). If the program has at least a feasible solution, then the optimal solutions are the ones that minimize the objective function (3.9a) which is a linear combination of variables of x weighted by the components of the vector c.

In the GED definition provided in Problem 5, the edit operations that are allowed to transform the graphs G_1 and G_2 are (i) the substitution of a vertex (respectively an edge) of G_1 with a vertex (resp. an edge) of G_2, (ii) the deletion of a vertex (or an edge) from G_1 and (iii) the insertion of a vertex (or an edge) in G_1. For each type of edit operation, we define a set of corresponding binary variables:

- $\forall (i,k) \in V_1 \times V_2$,
  $x_{i,k} = \begin{cases} 1 \text{ if } i \text{ is substituted with } k, \\ 0 \text{ otherwise.} \end{cases}$
- $\forall (ij,kl) \in E_1 \times E_2$,
  $z_{ij,kl} = \begin{cases} 1 \text{ if } ij \text{ is substituted with } kl, \\ 0 \text{ otherwise.} \end{cases}$
- $\forall i \in V_1, u_i = \begin{cases} 1 \text{ if } i \text{ is deleted from } G_1 \\ 0 \text{ otherwise.} \end{cases}$

- $\forall ij \in E_1, e_{ij} = \begin{cases} 1 \text{ if } ij \text{ is deleted from } G_1 \\ 0 \text{ otherwise.} \end{cases}$
- $\forall k \in V_2, v_k = \begin{cases} 1 \text{ if } k \text{ is inserted in } G_1 \\ 0 \text{ otherwise.} \end{cases}$
- $\forall kl \in E_2, f_{kl} = \begin{cases} 1 \text{ if } kl \text{ is inserted in } G_1 \\ 0 \text{ otherwise.} \end{cases}$

In order to evaluate the global cost of an edit path, elementary costs for each edit operation must be defined. We adopt the following notations for these costs:

- $\forall (i,k) \in V_1 \times V_2$, $c(i \to k)$ is the cost of substituting the vertex i with k,
- $\forall (ij,kl) \in E_1 \times E_2$, $c(ij \to kl)$ is the cost of substituting the edge ij with kl,
- $\forall i \in V_1$, $c(i \to \epsilon)$ is the cost of deleting the vertex i from G_1,
- $\forall ij \in E_1$, $c(ij \to \epsilon)$ is the cost of deleting the edge ij from G_1,
- $\forall k \in V_2$, $c(\epsilon \to k)$ is the cost of inserting the vertex k in G_1,
- $\forall kl \in E_2$, $c(\epsilon \to kl)$ is the cost of inserting the edge kl in G_1.

The objective function (3.10) is the overall cost induced by an edit path (x,z,u,v,e,f) that transforms a graph G_1 into a graph G_2. In order to get the graph edit distance between G_1 and G_2, this objective function must be minimized.

$$\begin{aligned} C(x,z,u,v,e,f) = \Bigg(& \sum_{i \in V_1} \sum_{k \in V_2} c(i \to k) \cdot x_{i,k} \\ & + \sum_{ij \in E_1} \sum_{kl \in E_2} c(ij \to kl) \cdot z_{ij,kl} \\ & + \sum_{i \in V_1} c(i \to \epsilon) \cdot u_i + \sum_{k \in V_2} c(\epsilon \to k) \cdot v_k \\ & + \sum_{ij \in E_1} c(ij \to \epsilon) \cdot e_{ij} + \sum_{kl \in E_2} c(\epsilon \to kl) \cdot f_{kl} \Bigg) \end{aligned} \tag{3.10}$$

Now let us present the constraints to guarantee that the admissible solutions of the BLP are edit paths that transform G_1 in G_2.

3.1.3.1.2.1 Vertices mapping constraints The constraint (3.11) ensures that each vertex of G_1 is either mapped to exactly one vertex of G_2 or deleted from G_1, while the constraint (3.12) ensures that each vertex of G_2 is either mapped to exactly one vertex of G_1 or inserted in G_1:

$$u_i + \sum_{k \in V_2} x_{i,k} = 1 \quad \forall i \in V_1 \tag{3.11}$$

$$v_k + \sum_{i \in V_1} x_{i,k} = 1 \quad \forall k \in V_2 \tag{3.12}$$

3.1.3.1.2.2 Edges mapping constraints Similarly to the vertex mapping constraints, the constraints (3.13) and (3.14) guarantee a valid mapping between the edges:

$$e_{ij} + \sum_{kl \in E_2} z_{ij,kl} = 1 \quad \forall ij \in E_1 \tag{3.13}$$

$$f_{kl} + \sum_{ij \in E_1} z_{ij,kl} = 1 \quad \forall kl \in E_2 \tag{3.14}$$

3.1.3.1.2.3 Topological constraints The respect of the graph topology in the mapping of the vertices and of the edges is described in the following proposition :

Proposition 3. *An edge $ij \in E_1$ can be mapped to an edge $kl \in E_2$ only if the head vertices $i \in V_1$ and $k \in V_2$, on the one hand, and if the tail vertices $j \in V_1$ and $l \in V_2$, on the other hand, are respectively mapped.*

This quadratic constraint can be expressed linearly with the following constraints (3.15) and (3.16):

- ij and kl can be mapped only if their head vertices are mapped:

$$z_{ij,kl} \leq x_{i,k} \quad \forall (ij,kl) \in E_1 \times E_2 \tag{3.15}$$

- ij and kl can be mapped only if their tail vertices are mapped:

$$z_{ij,kl} \leq x_{j,l} \quad \forall (ij,kl) \in E_1 \times E_2 \tag{3.16}$$

3.1.3.1.2.4 Reducing the number of constraints and variables The variables u, v, e and f help the reader to understand how the objective function and the constraints were obtained, but they are unnecessary to solve the GED problem.

Replacing in Equation 3.10 the variables u, v, e and f by their expressions deduced from equations 3.11,3.12, 3.13 and 3.14, we get a new objective function:

$$\begin{aligned} C'(x,z) = & \sum_{i \in V_1} \sum_{k \in V_2} \Big(c(i \to k) - c(i \to \epsilon) - c(\epsilon \to k) \Big) \cdot x_{i,k} \\ & + \sum_{ij \in E_1} \sum_{kl \in E_2} \Big(c(ij \to kl) - c(ij \to \epsilon) - c(\epsilon \to kl) \Big) \cdot z_{ij,kl} \\ & + \gamma \\ & \left(\text{with } \gamma = \sum_{i \in V_1} c(i \to \epsilon) + \sum_{k \in V_2} c(\epsilon \to k) + \sum_{ij \in E_1} c(ij \to \epsilon) + \sum_{kl \in E_2} c(\epsilon \to kl) \right) \end{aligned} \tag{3.17}$$

We transform the vertex mapping constraints 3.11 and 3.12 into inequality constraints, without changing their role in the program. As a side effect, it removes the u and v variables from the constraints:

$$\sum_{k \in V_2} x_{i,k} \leq 1 \quad \forall i \in V_1 \tag{3.18}$$

$$\sum_{i \in V_1} x_{i,k} \leq 1 \quad \forall k \in V_2 \tag{3.19}$$

We do the same for edge mapping constraints 3.13 and 3.14:

$$\sum_{kl \in E_2} z_{ij,kl} \leq 1 \quad \forall ij \in E_1 \tag{3.20}$$

$$\sum_{ij \in E_1} y_{ij,kl} \leq 1 \quad \forall kl \in E_2 \tag{3.21}$$

Equation 3.17 shows that the GED can be obtained without explicitly computing the variables u, v, e and f. Once the formulation solved, all insertion and deletion variables can be *a posteriori* deduced from the substitution variables.

The number of topological constraints, 3.15 and 3.16, is $|E_1| \cdot |E_2|$. Therefore, in average, the number of constraints grows quadratically with the density of the graphs. We show that it is possible to formulate the GED problem with potentially less constraints, leaving,the set of solutions unchanged. To this end, we propose to mathematically express Proposition 3 in another way. We replace the constraints 3.15 and 3.16 by the following ones:

- Given an edge $ij \in E_1$ and a vertex $k \in V_2$, there is at most one edge whose initial vertex is k that can be mapped with ij:

$$\sum_{kl \in E_2} z_{ij,kl} \leq x_{i,k} \quad \forall k \in V_2, \forall ij \in E_1 \tag{3.22}$$

- Given an edge $ij \in E_1$ and a vertex $l \in V_2$, there is at most one edge whose terminal vertex is l that can be mapped with ij:

$$\sum_{kl \in E_2} z_{ij,kl} \leq x_{j,l} \quad \forall l \in V_2, \forall ij \in E_1 \tag{3.23}$$

The entire formulation called F2 is described as follows :

Model 4. *F2*

$$\begin{aligned} \min_{x,z} \Bigg(& \sum_{i \in V_1} \sum_{k \in V_2} \Big(c(i \to k) - c(i \to \epsilon) - c(\epsilon \to k) \Big) \cdot x_{i,k} \\ + & \sum_{ij \in E_1} \sum_{kl \in E_2} \Big(c(ij \to kl) - c(ij \to \epsilon) - c(\epsilon \to kl) \Big) \cdot z_{ij,kl} \\ & \Bigg) + \gamma \end{aligned} \tag{3.24a}$$

$$\textit{subject to} \quad \sum_{k \in V_2} x_{i,k} \leq 1 \quad \forall i \in V_1 \tag{3.24b}$$

$$\sum_{i \in V_1} x_{i,k} \leq 1 \quad \forall k \in V_2 \tag{3.24c}$$

$$\sum_{kl \in E_2} z_{ij,kl} \leq x_{i,k} \quad \forall k \in V_2, \forall ij \in E_1 \tag{3.24d}$$

$$\sum_{kl \in E_2} z_{ij,kl} \leq x_{j,l} \quad \forall l \in V_2, \forall ij \in E_1 \tag{3.24e}$$

$$\textit{with} \quad x_{i,k} \in \{0,1\} \quad \forall (i,k) \in V_1 \times V_2 \tag{3.24f}$$

$$z_{ij,kl} \in \{0,1\} \quad \forall (ij,kl) \in E_1 \times E_2 \tag{3.24g}$$

$$\textit{where} \quad \gamma = \sum_{i \in V_1} c(i \to \epsilon) + \sum_{k \in V_2} c(\epsilon \to k) + \sum_{ij \in E_1} c(ij \to \epsilon) + \sum_{kl \in E_2} c(\epsilon \to kl) \tag{3.24h}$$

γ is not a function of x and z. It does not impact the minimization problem. However, γ is mandatory to obtain the GED value.

3.1.3.1.3 Exact method: Branch and Bound algorithm (DF) A part from mathematical models, A* is another exact method. A* is a tree-based method adopting a best-first exploration. A* stores candidate solutions to be explored while traversing the search tree. It is memory expensive, so we decided to investigate an algorithm with a memory efficient framework. To overcome the high memory load, we propose a depth-first graph edit distance (called DF) algorithm which requires less memory and search time. The search space is organized as an ordered tree which is explored in a depth-first way. Each tree node is a complete or a partial solution of the GED problem. For example, the first floor ($\mathcal{F}$) of the tree is obtained by creating a node for each substitution ($i \to k$, $\forall k \in V_2$) and a last tree node is added to the floor representing the deletion ($i \to \epsilon$). The choice of the most promising tree node is achieved by selecting the minimum cost tree node within the floor $\mathcal{F}$ (i.e., $p_{min} = arg \min_{p \in \mathcal{F}}(g(p) + h(p))$). $g(p)$ is the sum of the costs of the graph components in p. $h(p)$ is an estimation of the remaining cost of the unmatched graph components. After comparing several heuristics *h(p)* from the literature, we selected the bipartite graph matching heuristic proposed in [Riesen et al., 2007]. The complexity of such a method is $O(\max(|V_1|, |V_2|)^3)$. For each tree node p, the unmatched vertices and edges are handled completely independently. Unmatched vertices of G_1 and unmatched vertices of G_2 are matched at best by solving an linear sum assignment problem. Unmatched edges of both graphs are handled analogously. Obviously, this procedure allows multiple substitutions involving the same vertex or edge and, therefore, it possibly represents an invalid way to edit the remaining part of G_1 into the remaining part of G_2. However, the estimated cost certainly constitutes a lower bound of the optimal cost. Once p_{min} is selected then the child tree nodes are created by substituting ($j \to l$, $\forall l \in V_2 \setminus \{k\}$) and so on. While traversing the search tree, each leaf node is a feasible solution and provides an upper bound. The best found upper bound is saved in a variable called UB. A backtrack to the parent node is performed when a leaf node is reached. Finally, during the search, if $g(p) + h(p)$ is greater than UB then p is discarded. In the worst case, $|V_1|.|V_2|$ partial solutions to be explored are stored and hence the memory consumption is not exhausted. This method is called DF.

3.1.3.2 Deadlock 2: Performance evaluation of graph matching methods

3.1.3.2.1 Motivation The error-correcting graph matching problem is often evaluated in a classification context and less deeply assessed in terms of deviation to the optimal solution. So, we

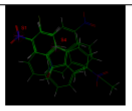
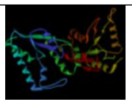
Database	Decomposition	Overview	Purpose
GREC	MIX, 5, 10, 15 and 20 vertices		Classification
MUTA	MIX, 10, 20, ... , 70 vertices		Classification
Protein	MIX, 20, 30 and 40 vertices		Classification
CMU	30 vertices		Matching

Figure 3.11: Graph data repository for error correcting graph matching

Database	#subsets	Max Graphs Size	#graphs per subset	#comparisons per subset	#optimal solutions
GREC	5	20	10	100	441
MUTA	8	70	10	100	189
Protein	5	40	10	100	47
CMU	1	30	111	660	128

Table 3.6: Overview about the subsets included in the repository

propose to provide publicly available data sets for assessing GED methods in term of deviation to the optimal solution or to the best known solution.

3.1.3.2.2 Graph data repository for error tolerant graph matching This proposal consists of three parts. First, we provide a graph database repository annotated with low level information like graph edit distances and their matching correspondences. Second, we propose a set of performance evaluation metrics to assess the performance of GED methods. Third, sub data sets were created considering the number of vertices of the graphs. The sub sets were designed to evaluate the behaviour of the methods as the number of vertices increases. The data bases are pictured out in Figure 3.11. For each graph pair, two ground truth elements are added: the distance between each pair of graphs and Vertex-to-Vertex matching. Theses values can come from an optimal solution or the best solution found so far among the methods that were put to the test. The number of optimal solutions is Tabulated in Table 3.6. Finally, a set of metrics was proposed. Especially, for a given data set, each method can be projected into a 2-Dimensional space corresponding to its average deviation and its average running time. Therefore, each method is a point in this space and a method is said to be dominated in the sens of Pareto if a method is not better than any other on one of the two criteria (average deviation and average running time).

The benchmark is available online http://www.rfai.li.univ-tours.fr/PublicData/GDR4GED/home.html. It is composed of metrics, sub sets, low level information (distance and matching) and well-defined cost functions.

3.1.3.2.3 ICPR 2016 - graph edit distance contest (GDC) This new benchmark gave birth to a contest during the ICPR conference in 2016. Its main challenge was to inspect and report performances and effectiveness of exact and approximate graph edit distance methods by comparison with a ground truth. The time constraint used in the contest was fixed to 30 s. That

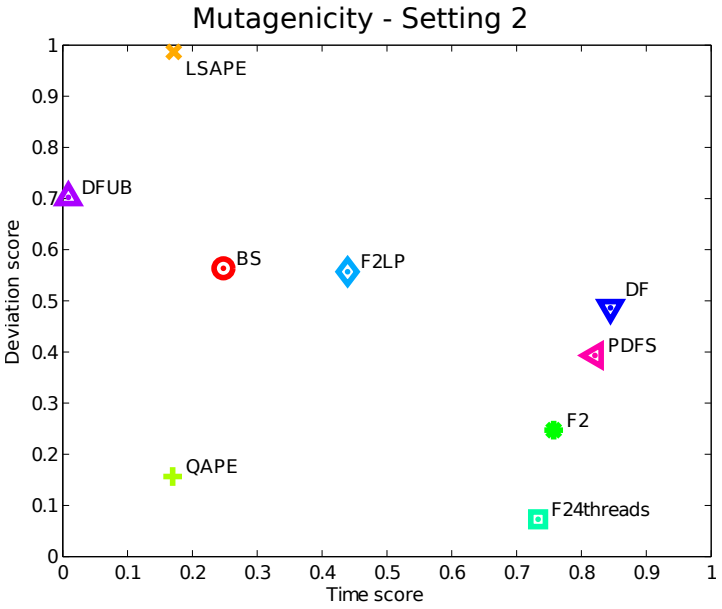


Figure 3.12: Average speed-deviation scores on MUTA Dataset.

is, the methods that needed more than 30 s were stopped, and the best answer found so far was outputted. Several data sets composed of graphs with symbolic and numeric attributes were used in the experiments. Table 3.7 summarizes the GED methods that were included in the contest.

Acronym	Reference	Details
BS-100	Neuhaus and Bunke. [2007]	Beam-search of size 100
LSAPE	Bougleux et al. [2017b]	Linear Sum Assignment Problem with Edition
QAPE=mIPFP	Bougleux et al. [2017a]	Quadratic Assignment Problem with Edition
F2	Lerouge et al. [2017]	Exact binary linear programming formulation
F24threads	Abu-Aisheh et al. [2017a]	Parallel version of *F2*
F2LP	Abu-Aisheh et al. [2017a]	Upper bound of *F2*
DF	Abu-Aisheh et al. [2015b]	Depth-first algorithm
DFUB	Abu-Aisheh et al. [2017a]	Upper bound of *DF*
PDFS	Abu-Aisheh et al. [2018]	Parallel version of *DF*

Table 3.7: Methods included in the graph edit distance contest of ICPR 2016.

For the sake of clarity, we synthesize our different conclusions via figures. For exhaustive and numerical results, we refer the interested reader to the contest website: http://gdc2016.greyc.fr. Results on MUTA and CMU House data sets are depicted in Figure 3.12 and Figure 3.13.

From the contest, an interesting remark came out. Time-truncated exact methods based on mathematical programming are really effective (accurate) while time-truncated exact methods based on a search tree can be fast (DFUB).

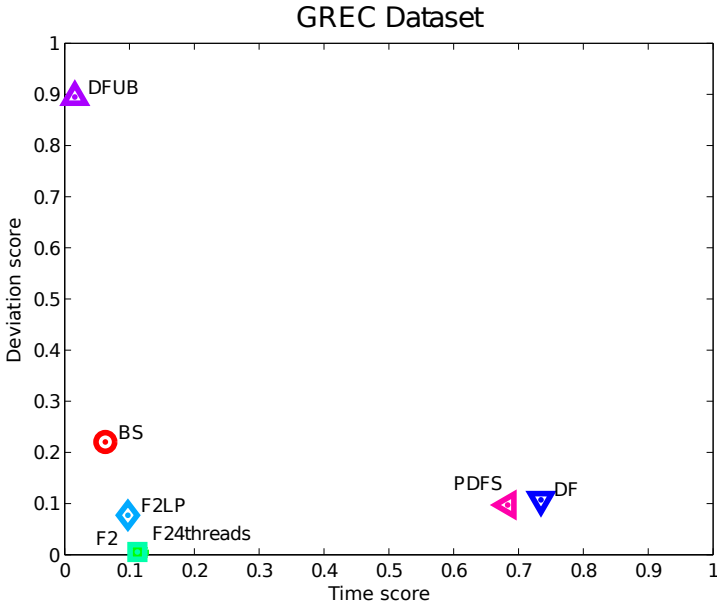


Figure 3.13: Average speed-deviation scores on GREC Dataset.

3.1.3.3 Deadlock 3: heuristics performance according to speed and effectiveness criteria

In this paragraph, we talk about accuracy and speed of heuristics. To create accurate heuristics, methods based on mathematical models are investigated. Seconds, to design fast and polyvalent methods, the modification of a tree-based method is studied.

3.1.3.3.1 Heuristics-based on a mathematical model

3.1.3.3.1.1 Motivation The effectiveness of the ILP formulations for the error-correcting graph matching was shown in [Lerouge et al., 2017]. Solving optimally the GED problem is limited to small-size instances. Lately, a new family of heuristics, namely matheuristics, has been introduced in Operation Research community. They involve both ILP formulations and solvers in a defined scheme with a mutual goal. The goal is to explore the solution space efficiently and compute very good quality solutions. One well-known matheuristics, called local branching, was introduced by [Fischetti and Lodi, 2003]. Its main idea is to perform a series of local searches in the solution space. The method focuses the search in defined regions looking for good quality solutions of an ILP formulation. Starting from an initial solution, it defines the neighborhood around it and performs an intensification step looking form better solutions. Local branching can deal with the problem of local optima by introducing a diversification step to escape them. This heuristic combines several techniques (neighborhood definition, intensification and diversification) in a branching scheme. Local branching was used to solve many optimization problems and has obtained very good results. Such a heuristic has not yet been tested on the GED problem in the literature. Furthermore, the diversification mechanism is modified to fit the GED problem.

3.1.3.3.1.2 Local branching details As presented in [Fischetti and Lodi, 2003], LocBra heuristic is a local search approach that makes use of ILP solver to explore the neighborhoods of solutions through a branching scheme. In addition, it involves mechanisms such as intensification

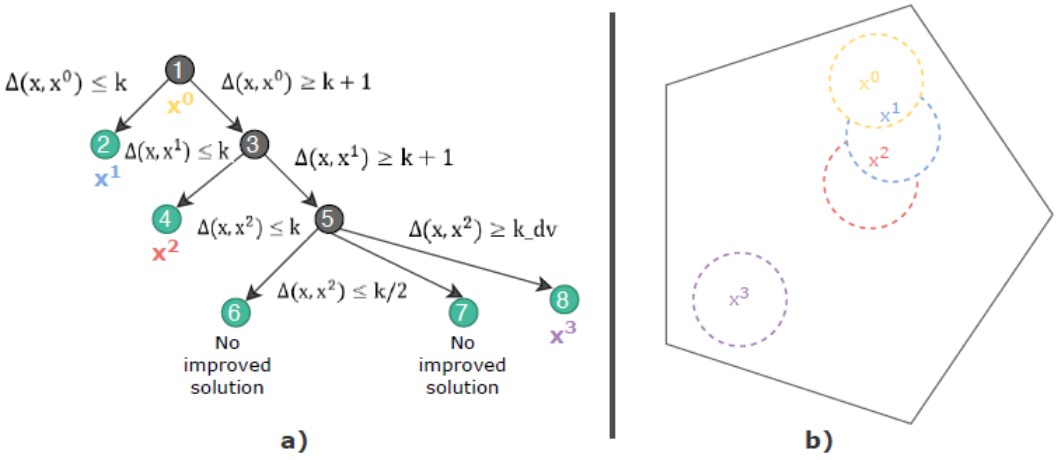


Figure 3.14: Local branching flow. a) depicts the left and right branching. b) shows the neighborhoods in the solution space

and diversification. Starting from an initial solution x^0, it defines the *k-opt neighborhood* $N(x^0, k)$, with k a given integer. In other words, the neighborhood set contains the solutions that are within a distance no more than k from x^0 (in the sense of *Hamming distance*). This implies adding the following *local branching constraint* to the ILP model:

$$\Delta(x, x^0) = \sum_{j \in S^0} (1 - x_j) + \sum_{j \in B \setminus S^0} x_j \leq k \tag{3.25}$$

such that, B is the index set of binary variables defined in the model, and $S^0 = \{j \in B : x_j^0 = 1\}$. Δ is a Hamming distance between two feasible solutions x and x^0. The basis of LocBra is illustrated in Figure 3.14.

The original version of local branching is designed to solve any optimization problem with existing ILP formulations. Of course, integrating GED properties and information about the instance in the heuristic will help in improving its performance. The improvements are integrated by adapting certain mechanisms of the method. The first particularization relates to the choice of the variables for defining the search space. Traditionally, in a local branching heuristic all boolean variables are considered to define the local branching constraint Equation 3.25. For the GED problem, it turns out that the crucial variables are x that model the vertices matching. Another important improvement is proposed for the diversification mechanism, where also not all binary variables are included but a smaller set of "important variables" is used instead. The notion of important variables is based on the idea that when changing its value from $1 \rightarrow 0$ (or the opposite), it highly impacts the objective function value.

3.1.3.3.2 Anytime Branch and Bound (ADF)

3.1.3.3.2.1 Motivation We establish a compromise between exact and approximate error-correcting GM algorithms, referred to here as anytime algorithms. The concept of anytime algorithms was first reported in [Zilberstein and Russell, 1995]. The desirable properties of anytime algorithms are as follows:

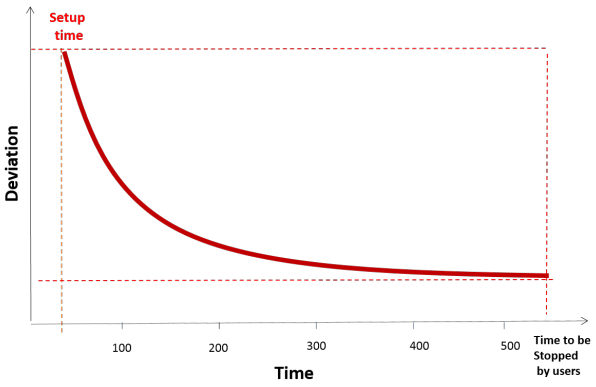


Figure 3.15: Characteristics of anytime algorithms

- Interruptibility: After some small amount of setup time[3], a suboptimal solution can be provided by stopping the algorithm at time t.
- Monotonicity: The quality of the result increases as a function of computational time.
- Measurable quality: We can always measure the quality of a suboptimal result.
- Preemptability: Anytime algorithms can be suspended and resumed with minimal overhead.

Anytime algorithms have a trade-off between quality and execution time, see Figure 3.15. They can find the first best-so-far solution after some setup time at the beginning of the execution. From Figure 3.15, one can see that the quality of the solution improves with increasing execution time. Users have the choice of stopping the algorithm at anytime and thus getting an answer that is satisfactory, or they can run their algorithm until its completion when it is important to find the optimal solution.

3.1.3.3.2.2 Details The anytime algorithm for GED is named ADF and it is based on DF. During traversal of the search tree when a first complete solution is reached, it is outputted and made available for the application or the user. Thereafter, when a better solution is found (a better upper bound UB) then it is outputted and made available too. This process is repeated until the optimal solution is found or the user stops the method. ADF can start from a pre-computed upper bound. In such a case, ADF is called $ADF - UB$.

Figure 3.16 depicts the list of improved solutions (i.e., distances) found by the anytime methods ADF-UB and ADF on one random pair of graphs taken from MUTA. One can observe that in the first few milliseconds ADF-UB does not output any solution while ADF succeeds in outputting several solutions, however, with the delight of time both of them reach the same distance. Such a fact reveals the importance of anytime algorithms, since they are able to improve their solutions in a few milliseconds.

Whenever an improved solution is found, it is made available for the final application that uses the anytime method as a service. This way of understanding the problem makes the anytime method very flexible and applicable when the time constraints of the final application are not known in advance.

[3]The time needed to output a first solution by an anytime method.

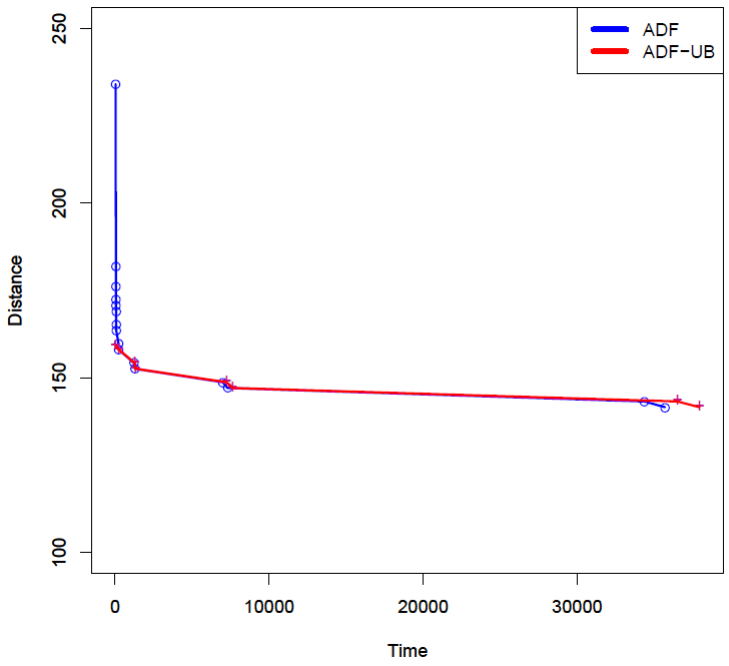


Figure 3.16: A random pair of graphs taken from MUTA-70 database illustrating the improvement of found solutions with the delight of time

3.1.3.3.2.3 Deviation evolution through time In the graph edit distance contest, a time limit was set to stop methods after a certain time (for instance 30s). However, to provide a better understanding of the algorithms it is important to consider a time analysis at different time steps.

Figure 3.17 pictures out results on MUTA-70 that is a challenging data set with graphs of 70 nodes. The left part of Figure 3.17 shows that when time matters, *FBP* was the fastest in outputting solutions, followed by *BP*, *SBP-Beam* and *ADF-UB*. After 40 ms (right of Figure 3.17), both *ADF* and *ADF-UB* beat *BP*. For instance, when the time limit was equal to 400 ms, the deviation of *BP* was 45.24% whereas the deviation of *ADF* and *ADF-UB* was 35.12% and 33.02%, respectively.

3.1.3.4 Deadlock 4: Relation between Problem ETSGM and Problem ECGM

In this paragraph, we propose to draw a tighter relation between both problems. Especially, we create a link between both models Model GMIQP and Model SGMIQP. Our proposition goes as follows:

Proposition 4. *Model GMIQP and Model SGMIQP are equivalent in terms of solutions under a reformulation of the cost function* $s(i \to k) = -\left(c(i \to k) - c(i \to \epsilon) - c(\epsilon \to k)\right)$.

To intuitively demonstrate the exactness of the proposition, we proceed as follows :

1. We start from Model GMIQP.
2. From this quadratic model, we express the Model F2 which is a linear model.
3. We perform a change of cost functions linking s with c.

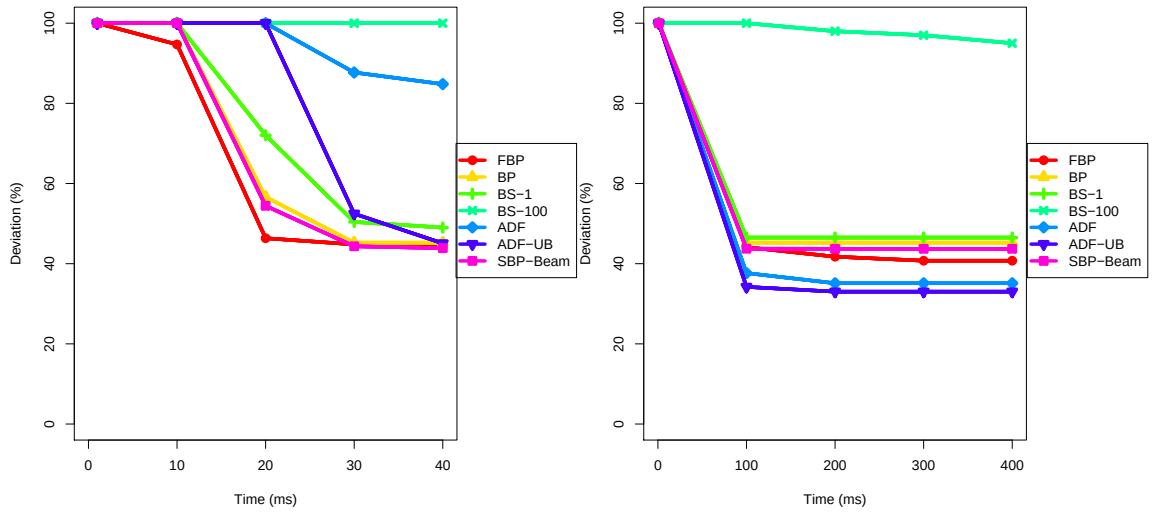


Figure 3.17: MUTA deviation: left (up to 40 ms), right (up to 400 ms).

4. Under this cost function s, we show that F2 turns to be a maximization problem and we call this new model F2'.
5. F2' is then modify to get a quadratic model called GMIQP'.
6. We obtain GMIQP' = Model SGMIQP and it is sufficient to show that both models solve the same problem.

Sketch of proof. Let Γ_1 be the set of solutions (edit paths between G_1 and G_2) implied by the set of admissible solutions of the Model GMIQP. Let Γ_2 be the set of solutions implied by the set of admissible solutions of F2 (Model 4).

1. Model GMIQP and Model F2 solve the error-correcting graph matching problem so $\Gamma_1 = \Gamma_2$.
2. By setting $d(i \to k) = (c(i \to k) - c(i \to \epsilon) - c(\epsilon \to k))$ and $d(ij \to kl) = (c(ij \to kl) - c(ij \to \epsilon) - c(\epsilon \to kl))$, we can rewrite the objective function of F2 as follows :

$$C'(x,z) = \sum_{i \in V_1} \sum_{k \in V_2} d(i \to k) \cdot x_{i,k} + \sum_{ij \in E_1} \sum_{kl \in E_2} d(ij \to kl) \cdot z_{ij,kl} + \gamma$$
$$\left(\text{with } \gamma = \sum_{i \in V_1} c(i \to \epsilon) + \sum_{k \in V_2} c(\epsilon \to k) + \sum_{ij \in E_1} c(ij \to \epsilon) + \sum_{kl \in E_2} c(\epsilon \to kl) \right) \tag{3.26}$$

3. γ does not depend on variables so it does not impact the optimization problem. Therefore γ can be removed.
4. By setting $s(i \to k) = -d(i \to k) = -(c(i \to k) - c(i \to \epsilon) - c(\epsilon \to k))$ and $s(ij \to kl) = -d(ij \to kl)$, we can rewrite the objective function C' of the model F2 to obtain C''.

$$C''(x,z) = \sum_{i \in V_1} \sum_{k \in V_2} s(i \to k) \cdot x_{i,k} + \sum_{ij \in E_1} \sum_{kl \in E_2} s(ij \to kl) \cdot z_{ij,kl} \tag{3.27}$$

5. Minimizing $f(x)$ is equivalent to maximize -$f(x)$. So, minimizing C' is equivalent to maximize C''.

6. The linear objective function C'' can be turned into a quadratic function by removing variables z and replacing them by product of x variables.

$$C'''(\mathbf{x}) = \sum_{i \in V_1} \sum_{k \in V_2} s(i \to k) \cdot x_{i,k} + \sum_{ij \in E_1} \sum_{kl \in E_2} s(ij \to kl) \cdot x_{i,k} \cdot x_{j,l} \tag{3.28}$$

7. Topological constraints (Equations 3.24d and 3.24e (see below)) in F2 are not necessary anymore and they can be removed. The product of $x_{i,k}$ and $x_{j,l}$ is enough to ensure that ab edge $ij \in E_1$ can be matched to an edge $kl \in E_2$ only if the head vertices $i \in V_1$ and $k \in V_2$, on the one hand, and if the tail vertices $j \in V_1$ and $l \in V_2$, on the other hand, are respectively matched.

$$\sum_{kl \in E_2} z_{ij,kl} \leq x_{i,k} \quad \forall k \in V_2, \forall ij \in E_1$$
$$\sum_{kl \in E_2} z_{ij,kl} \leq x_{j,l} \quad \forall l \in V_2, \forall ij \in E_1$$

8. We obtain the new model named GMIQP':

Model 5. *GMIQP'*

$$\max_{\mathbf{x}} C''' \tag{3.29a}$$

$$\textit{subject to} \quad \sum_{k \in V_2} x_{i,k} \leq 1 \quad \forall i \in V_1 \tag{3.29b}$$
$$\sum_{i \in V_1} x_{i,k} \leq 1 \quad \forall k \in V_2 \tag{3.29c}$$
$$\textit{with} \quad x_{i,k} \in \{0,1\} \quad \forall (i,k) \in V_1 \times V_2 \tag{3.29d}$$
$$\tag{3.29e}$$

9. Model SGMIQP = Model GMIQP'. This was to be demonstrated. Proposition 4 is right.

□

Under condition of Proposition 4, the optimal assignment obtains when solving Model SGMIQP allows to reconstruct an optimal solution for the Model GMIQP and to compute the associated GED. This process is depicted in Figure 3.18. Note that in some case, the function $s()$ has to be positive then $s()$ can rewrite $s(i \to k) = cst - (c(i \to k) - c(i \to \epsilon) - c(\epsilon \to k))$ with cst a large constant value.

Proposition 4 is a first attempt toward the unification of two communities working respectively on GED and subgraph matching problems. The proposition needs to be validated and fully proved with more examples and numerical experiments. All the methods solving the Problem ETSGM represented by Model SGMIQP can be used to solve the Problem ECGM under a specific cost function $s(i \to k) = -(c(i \to k) - c(i \to \epsilon) - c(\epsilon \to k))$.

Operation	Cost	Similarity
i → k	0	-1.(0-10-10)=20
i → l	10	-1.(10-10-30)=30
$\varepsilon_l \rightarrow l$	30	NA
$\varepsilon_k \rightarrow k$	10	NA
$i \rightarrow \varepsilon_i$	10	NA

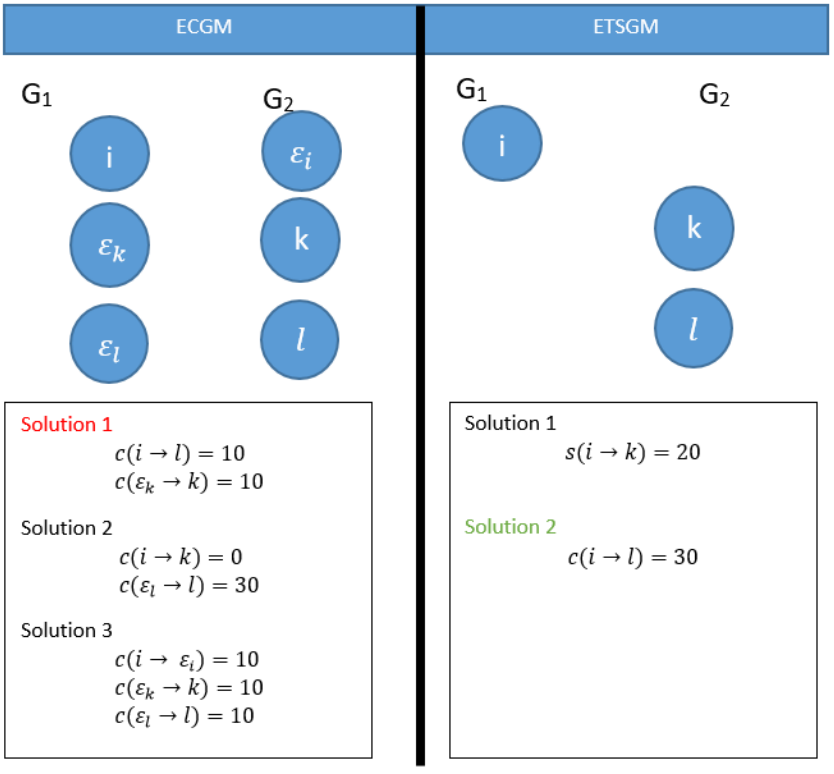


Figure 3.18: A comparison of the sugbraph matching and error-correcting graph matching problems when the similarity function $s(i \rightarrow k) = -\{c(i \rightarrow k) - c(i \rightarrow \epsilon) - c(\epsilon \rightarrow k)\}$

3.1.4 Summary

In the PhD thesis of Zeina Abu-Aisheh [Abu-Aisheh, 2016], an exact graph matching method was developed based on a memory efficient Branch and Bound technique [Abu-Aisheh et al., 2015b]. This method was turned into an anytime algorithm to cope with uncertainty of the time required by the final applications [Abu-Aisheh et al., 2016]. To gain in efficiency, a parallel version was designed [Abu-Aisheh et al., 2018]. A benchmark was made by annotating existing data sets with low level information (distance, vertex matching) and proposing new metrics [Abu-Aisheh et al., 2015a]. A ILP model was proposed in [Lerouge et al., 2017]. An exact method based on this mathematical model was designed [Lerouge et al., 2016] along with a time-truncated heuristic [Lerouge et al., 2017]. All these methods were put to the test in the ICPR 2016 graph edit distance contest [Abu-Aisheh et al., 2017a]. In the PhD thesis of Mostafa Darwiche, an heuristic based on mathematical models was proposed [Darwiche et al., 2018, in pressa]. An attempt was made to link researchers working on error-tolerant subgraph matching and error-correcting graph matching. Our codes are available at:

- Anytime Graph Matching `http://www.rfai.li.univ-tours.fr/PublicData/GraphLib/home.html`
- Local Branching `https://sites.google.com/view/orspr/`

3.2 Graph classification

This section is decomposed into three parts: Section 3.2.1, the state of the art where problems are expressed along with the methods from the literature. Section 3.2.2, the deadlocks to be released are stated about the combinatorial aspect of the graph classification. Section 3.2.3, answers and analysis of the open problems are proposed.

3.2.1 State of the art

Graph classifiers can be categorized into two categories whether the classifier operates in graph space or in vector space. In this manuscript, we only focus on methods operating in graph space without projecting the graphs into a vector space. This strong assumption is motivated by the aim of capturing as much as possible structural distortions. Graph matching is a way to compare graphs in graph space and a standard approach to GED-based pattern recognition is given by the k-nearest neighbors (kNN) classification [Serratosa, 2019, Riesen, 2015, Serratosa, 2015, Cortés and Serratosa, 2015]. In contrast with other classifiers such as artificial neural networks, Bayes classifiers, or decision trees, the underlying pattern space need not be rich in mathematical operations for nearest-neighbor classifiers to be applicable. In this scenario, a test graph from the test set (TeS) is compared to all the graphs in the training set with the aim of defining a neighborhood based on a dissimilarity measure between graphs. Finally, the test graph is assigned to the most common class among its neighborhood.

3.2.1.1 Problem definition

Let us recall some notations. Let $\mathcal{D}$ be the set of graphs and let $\mathcal{T}$ be the set of classes. Given a graph training set $TrS = \{(G_j, t_j)\}_{j=1}^{M}$, where $G_j \in \mathcal{D}$ is a graph and $t_j \in \mathcal{T}$ is the class of the graph.

The 1-nearest neighbor problem can be defined as follows:

Problem 10. *1-Nearest Neighbor Problem (1NN)*

$$(G^*, t^*) = arg \min_{(G_j,t_j)\in TrS} d(G, G_j) \tag{3.30}$$

Where $d(G, G_j)$ is an arbitrary function to calculate a dissimilarity between G and G_j.

To extend Problem 10 to k-nearest neighbors, we introduce $\mathcal{K}$, the set of the kNN from a query graph $G \in TeS$. Let $\mathcal{K} = \{(G_1, t_1), \cdots, (G_j, t_j), \cdots, (G_k, t_k)\}$ be a set of graphs along with their class labels with $(G_j, t_j) \in TrS$. The k-Nearest Neighbors problem can be defined by:

Problem 11. *k-Nearest Neighbors Problem (kNN)*

$$\mathcal{K} = arg \underset{(G_j,t_j)\in TrS}{sort} (d(G, G_j), k) \tag{3.31}$$

Where sort is a function that performs an ascending sort of $d(G_i, G_j)$ values. k is the number of retained values to choose the number of nearest neighbors of G.

Complexity of Problem 11 is $O(M\sigma)$ with σ the complexity of the dissimilarity function. In the literature, the complexity of the dissimilarity function is often ignored to focus on the complexity of the kNN.

To exploit the Problem 11 in a classification context, a voting operator has to be defined. The max voting operator is a function $\rho : \mathcal{K} \to \mathcal{T}$ defined by:

Definition 10. *Max Voting Operator:* ρ

$$t_j^* = arg\max_{t_j \in \mathcal{T}} count(\mathcal{K}, t_j) \tag{3.32}$$

Where $count(.)$ *is a function that counts the number of observations that fall into each class* t_j.

Problem 10 and Problem 11 put forward that $d(G, G_j)$ is computed between G and each graph G_j in the training set.

The optimal kNN can be found by an exact kNN algorithm when :

1. Solving $d(G, G_j)$ by an exact graph matching method,
2. and comparing G with all the training set.

The kNN classifier has many advantages thanks to three properties: i) It is non-parametric. It means that it does not depend on a specific distribution of the data (i.e. Gaussian distribution). ii) Only one parameter k is needed. iii) The kNN classifier is intuitive. It means that a wrong or a good decision can be easily understood by a human by looking at the set of nearest neighbors. However, its time consumption cannot be ignored especially when the number of graphs in the training set is big.

The first use of the kNN classifier is reported in [Cover and Hart, 1967]. It is stated that the probability of error of the nearest neighbor rule is bounded above by twice the Bayes probability of error. In this sense, it may be said that half the classification information in an infinite sample set is contained in the nearest neighbor. In its origin the kNN problem is a type of lazy learning algorithms such that training is not needed. In machine learning, this kind of method is referred to instance-based learning. The parameter k has an important impact on the classification rate of kNN. This impact was deeply studied in [Batista and Silva, 2009]. The boundary between classes becomes smoother with increasing value of k. A small value of k (i.e $k = 1$) could lead to an over-fitting phenomena while a large value of k could lead to under-fitting of the data (under-fitting and over-fitting are defined in Appendix B). k is usually empirically tuned by checking the error rate on a validation data set.

3.2.1.2 kNN methods

The literature about the kNN algorithms can be split into two parts : exact or approximate nearest neighbor search. Does a method return the optimal nearest neighbors or not? The quality and usefulness of the algorithms are determined by the time complexity of queries as well as the space complexity of any search data structures that must be maintained. A survey of nearest neighbor techniques can be found in [Bhatia and Vandana, 2010].

3.2.1.2.1 Exact methods

3.2.1.2.1.1 Linear search (structure less) The first method is called "Linear search". The simplest solution to the kNN problem is to compute the distance from the query to every other data in the database, keeping track of the "best so far". This algorithm, sometimes referred to as the naive approach, has a running time of $O(M)$[4] where M is the cardinality of $\mathcal{D}$. There are no search data structures to maintain, so linear search has no space complexity beyond the storage of the database.

[4] The complexity is given regardless to the dissimilarity function complexity.

3.2.1.2.1.2 Space partitioning Several space-partitioning methods have been developed for solving the kNN problem. Perhaps the simplest is the k-d tree, which iteratively bisects the search space into two regions containing half of the points of the parent region. Queries are performed via traversal of the tree from the root to a leaf by evaluating the query point at each split. k-d-trees are restricted to euclidean space. A limitation of these multidimensional search structures is that they are only defined for searching over objects that can be treated as vectors. They aren't applicable for the more general case in which the algorithm is given only a collection of objects and a function for measuring the distance or similarity between two objects. However in case of general metric space, branch and bound approaches are still applicable and thy are known under the name of metric trees. The first use of the term "metric tree" was published in [Uhlmann, 1991]. Particular examples include vp-tree and BK-tree [Yianilos, 1993]. However, the dissimilarity function must satisfy the **triangle inequality** then the result of each comparison can be used to prune the set of graph candidates to be examined. The triangle inequality allows to compute bounds on various distances without having to evaluate the distance function itself [Kumar et al., 2008]. The graph edit distance can satisfy the triangle inequality property if the graph edit distance is turned into a distance. To do so, the cost function associated with the edit operations satisfies the distance conditions of non-negativity, symmetry, and the triangle inequality then the GED is a distance [Neuhaus and Bunke., 2007] that respects the four axioms:

Definition 11. *Distance function*
A distance function $d : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ *is a distance if for all graphs* (G_i,G_j) *the four axioms are respected:*

- *Non-negativity:* $d(G_i, G_j) \geq 0$
- *Identity:* $d(G_i, G_j) = 0 \implies G_i = G_j$
- *Symmetry:* $d(G_i, G_j) = d(G_j, G_i)$
- *Triangle inequality:* $d(G_i, G_j) \leq d(G_i, G_k) + d(G_k, G_j)$

The bounding of distances thanks to the triangle inequality is illustrated by the use case of Figure 3.19. Considering the four graphs G_1, G_2, G_3, G_4, the first axiom implies that $d(G_1, G_4) \geq 0$ and $d(G_4, G_1) \geq 0$. The third axiom implies that $d(G_1, G_4) = d(G_4, G_1)$. The fourth axiom implies that :

$$d(G_1, G_4) \leq d(G_1, G_2) + d(G_2, G_4) \implies d(G_1, G_4) \leq 2 \tag{3.33a}$$
$$d(G_1, G_3) \leq d(G_1, G_4) + d(G_4, G_3) \implies d(G_1, G_4) \geq d(G_1, G_3) - d(G_4, G_3) \tag{3.33b}$$

So, we can conclude that $\sqrt{5} - 1 \leq d(G_1, G_4) \leq 2$. A method based on a metric tree can use these bounds to prune the search space.

3.2.1.2.2 Heuristic methods

3.2.1.2.2.1 Greedy search in proximity neighborhood graphs Proximity graph methods (such as HNSW[Malkov and Yashunin, 2016]) are considered as the current state-of-the-art for the approximate nearest neighbors search. The methods are based on greedy traversing in proximity neighborhood graphs $G = (V, E)$ in which every data $G_i \in \mathcal{D}$ is uniquely associated with vertex $v_i \in V$. The search for the nearest neighbors to a query q in the set $\mathcal{D}$ takes the form of searching for a specific vertex in the graph $G = (V, E)$. The basic algorithm is a *greedy search* and it works as follows: the search starts from an enter-data vertex $v_i \in V$ by computing the distances from the query q to each vertex of its neighborhood $\{v_j | (v_i, v_j) \in E\}$, and then finds a vertex with the minimal distance value. If the distance value between the query and the selected vertex is smaller

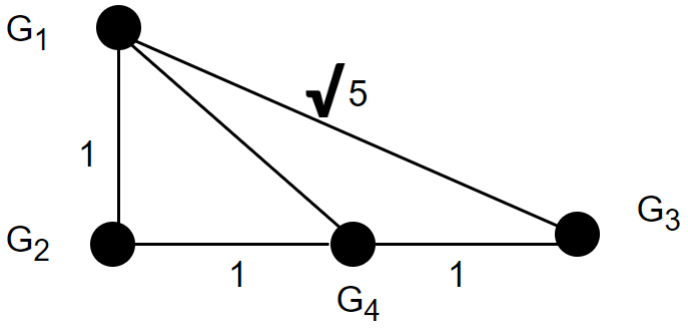


Figure 3.19: Four graphs and the distance between each graph.

than the one between the query and the current element, then the algorithm moves to the selected vertex, and it becomes new enter-data. The algorithm stops when it reaches a local minimum: a vertex whose neighborhood does not contain a vertex that is closer to the query than the vertex itself.

3.2.1.2.2.2 Locality sensitive hashing Locality sensitive hashing (LSH) is a technique for grouping points in space into 'buckets' based on some distance metric operating on the data. Data that are close to each other under the chosen metric are mapped to the same bucket with high probability [Rajaraman and Ullman, 2011].

3.2.1.2.2.3 Compression/clustering based search The database space can be cluster or structured to avoid comparing a test graph with all the graphs in the training set. It can be achieved by clustering algorithm operating in dissimilarity space such as Partition Around Medoids (PAM) [Kaufman and Rousseeuw, 1987] or hierarchical cluster analysis. More specifically, some methods have been adapted to graphs [Musmanno and Ribeiro, 2016, Chaieb et al., 2017, Ferrer et al., 2010, 2009, 2011, Borzeshi et al., 2013]. Globally, the aim is to construct prototypes that are representatives for $\mathcal{D}$. Clearly, this part is more related to the machine learning field and it will be developed in Section 4. One of the main limitations of such methods is the loss of information as they imply a significant reduction of the training set (with the use of the representatives of clusters instead of all the labeled samples). Boundaries between classes can then become less precise.

3.2.1.2.2.4 Fast heuristics for the GED problem Another approach to speed up the kNN computation is to find a fast (heuristic) GED algorithm. In Section 3.1, a review of heuristics for the GED problem have been proposed. Methods [Riesen and Bunke, 2009, Serratosa, 2015, Neuhaus et al., 2006, Riesen, 2015] have been applied to the kNN problem.

3.2.1.3 Analysis

To summarize, all the methods are tabulated in Table 3.8 according to the following criteria :

- Exact: Is the method exact or not ?
- Category : The family of the methods: linear search or space partitioning methods.
- Generic: Is the method generic operating in a metric space or is it adapted to graph space ?
- Learning: Does the method require a learning phase or not?

Method	Exact	Category	Generic	Learning
Brute force search [Cover and Hart, 1967]	yes	Line search	yes	no
Metric tree [Uhlmann, 1991]	yes	Space Partition	yes	no
Vp-tree [Yianilos, 1993]	yes	Space Partition	yes	no
Lower bound kNN [Kumar et al., 2008]	yes	Space Partition	yes	no
HNSW[Malkov and Yashunin, 2016]	no	Proximity graph	yes	yes
LSH [Rajaraman and Ullman, 2011]	no	Hash table	yes	yes
Clustering-kNN [Kaufman and Rousseeuw, 1987]	no	Clustering	yes	yes
Prototype-kNN [Musmanno and Ribeiro, 2016, Chaieb et al., 2017, Ferrer et al., 2010, 2009, 2011, Borzeshi et al., 2013]	no	Line search	no	yes
FastGED-kNN [Riesen and Bunke, 2009, Serratosa, 2015, Neuhaus et al., 2006, Riesen, 2015, Abu-Aisheh et al., 2018]	no	Prototype	no	no

Table 3.8: A summary of kNN methods.

	Generic	Graph oriented methods
Dissimilarity function	Metric property [Uhlmann, 1991, IPL]	Fast GED methods [Riesen, 2015, ACVPR]
Dissimilarity space	Space partition, prototypes, hashtable, proximity graph, line search [Malkov and Yashunin, 2016, PAMI]	Graph prototypes [Musmanno and Ribeiro, 2016, EJOR]

Figure 3.20: Speeding up the kNN problem : State of the art of kNN methods operating in dissimilarity space.

In Figure 3.20 methods are synthesized according to either methods are generic or dedicated to graphs. From Table 3.8 and Figure 3.20, several remarks can be drawn. Learning-free methods based on metric trees (i.e. vp-tree) imposes that the triangle inequality property holds true for the distance function. This constraint is very conservative and may lead to low recognition rates and a small speed up in practice [Serratosa, 2019]. Structuring the search space lies at the art of machine learning techniques and many kNN methods require a learning phase to structure the database. In the meantime, many kNN methods are learning-free and rely on fast GED solvers.

From these three remarks, we can observe the complementary of discrete optimization and machine learning. The two aspects can be combined to achieve a common goal : a good kNN method. The organization of the search space by means of machine learning techniques and the design of efficient graph matching solvers thanks to combinatorial optimization.

3.2.2 Open problems

From the literature review and focusing on learning-free methods, the methods are divided in two parts: i) the methods based on fast GED heuristics and ii) the methods based on a line search operating on a generic dissimilarity space. Speaking about the first category, fast GED heuristics are key elements of kNN methods but there is no clear conclusion in the literature about the impact of the suboptimality on the kNN methods. The same observation can be drawn about the classifiers based on kNN methods. On the other category of methods, they are based on a linear search among all the graph in the database. The line search methods do not take advantage of the graph space to structure the kNN search and so to speed up the methods. From these conclusions, three open problems can be drawn:

1. What is the impact of GED heuristics on the kNN problem?
2. Is there a way to specialize a line search method to operate in graph space instead of the generic dissimilarity space?
3. What is the impact of GED heuristics in a classification context?

3.2.3 Contribution

This section is built on the three open problems mentioned in Section 3.2.2.

3.2.3.1 Deadlock 5: What is the impact of GED heuristics on the kNN problem?

An important question is brought up: what is the impact of GED heuristics on the (dis)similarity search? This question can be answered experimentally by evaluating the ranking of graphs, which is considered as an important task in graph retrieval. A query graph is compared to each graph

of a database thanks to a given GED heuristic used as a distance. Then, distances are sorted in ascending order to obtain a ranking. The goal of this experiment is to compare the ranking given by all the heuristics against the optimal ranking given by an exact method. Consequently, the experiment aims at comparing the orders provided by the heuristics and the order provided by an exact method. It proceeds as follows: assuming a graph database with 5 graphs $\mathcal{D} = \{G_1, G_2, \cdots, G_5\}$. Starting with graph G_1, the optimal and heuristics solutions (distances) are computed for all possible pairs of graphs e.g. $d(G_1, G_1); d(G_1, G_2); \cdots; d(G_1, G_5)$. Then, graphs are ordered by ascending order based on the distances. For instance, assuming that the optimal order for G_1 is $O_1^{opt} = \{G_1, G_2, G_4, G_3, G_5\}$ and a heuristic pm order is $O_1^{pm} = \{G_1, G_5, G_4, G_3, G_2\}$. Then, the metric used is the Kendall rank correlation coefficient τ_b: it is a statistic used to study the correlation between two ranked/sorted ordinal variables [Kendall, 1948]. Computing τ_b consists in measuring the degree of concordance between the two ranked variables. The correlation τ_b is computed between O_1^{opt} and O_1^{pm} . The τ_b statistic makes adjustments for ties. Ties means that data that have the same value. For instance, the pair $((G_1, G_1), (G_1, G_2))$ is tied if $d(G_1, G_1) = d(G_1, G_2)$. A tied pair is neither concordant nor discordant. When tied pairs arise in the data, the coefficient may be modified in a number of ways to keep it in the range $[-1, 1]$. Values of τ_b range from -1 (100% negative association, or perfect inversion) to $+1$ (100% positive association, or perfect agreement). A value of zero indicates the absence of association. The Kendall τ_b coefficient is defined as:

$$\tau_b = \frac{n_c - n_d}{\sqrt{(n_0 - n_1)(n_0 - n_2)}}$$

$$n_0 = n(n-1)/2$$

$$n_1 = \sum_i t_i(t_i - 1)/2$$

$$n_2 = \sum_j u_j(u_j - 1)/2$$

where

$n_c =$ Number of concordant pairs

$n_d =$ Number of discordant pairs

$t_i =$ Number of tied values in the i^{th} group of ties for the first quantity

$u_j =$ Number of tied values in the j^{th} group of ties for the second quantity

By repeating this procedure for each graph of the database, $|\mathcal{D}|$ values of τ_b are computed for an heuristic pm.

For a given heuristic (pm), it is interesting to analyze the distribution of τ_b values computed for different graph query. Such a goal can be achieved by a statistical test. The hypothesis to be tested is $H0$: two variables O_1^{opt} and O_1^{pm} are not correlated and $\tau_b = 0$. The alternative hypothesis ($H1$) is that the variables are correlated, and τ_b is non-zero. A p-value test is applied to evaluate these hypothesis. The p-value is defined as the probability of obtaining a result equal to or "more extreme" than what was actually observed, when the null hypothesis is true. It is the error of the second kind, to accept the null hypothesis while H1 is true (false positive). If the p-value is less than a chosen risk level (5% for example), then the null hypothesis (H0) is rejected. On the contrary, if the p-value is greater than the chosen alpha level, then the null hypothesis (H0) cannot be rejected. In any case, it never leads to accept H1. A p-value is computed for each τ_b value, it represents the probability of obtaining results similar or better to what was observed if the null hypothesis is true. For each p-value under 5%, we can say that the hypothesis H0 is rejected.

The MUTA subset with graph size 30 is picked in this experiment, because all optimal solutions are known for these instances (100 instances) and 30 is the average graph sizes. All PAH instances (8836) are selected to be part of this experiment as well.

Method	Percentage
LocBra	100
SBPBeam	14
IPFP	21
GNCCP	2

Table 3.9: Percentage of the number of times, the H0 hypothesis ($\tau_b = 0$) was rejected for each heuristic on PAH instances

Method	Percentage
LocBra	100
SBPBeam	50
IPFP	50
GNCCP	70

Table 3.10: Percentage of the number of times, the H0 hypothesis ($\tau_b = 0$) was rejected for each heuristic on MUTA-30 instances

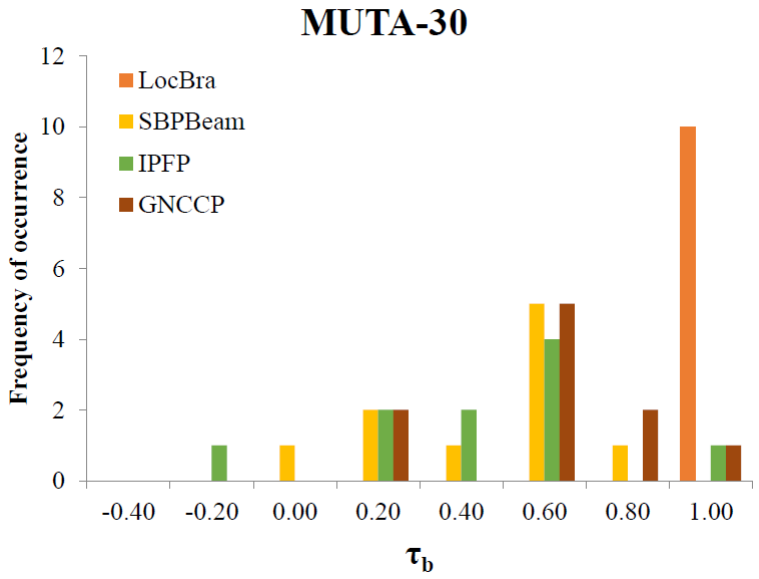
In Table 3.9, the percentage p-value under 5% is reported for the PAH data set. Local Branching heuristic (**LocBra**) is at 100%, so the null hypothesis is always rejected for all instances. LocBra has a very strong correlation with the optimal ranking. The other heuristics have lower percentages and are far from LocBra, the highest (23%) being obtained by GNCCP. In Figure 3.21 chart(b) shows τ_b distribution for PAH instances. LocBra has correlation values between [0.6; 1] and all the other heuristics are below those values. This proves that the ranking obtained by LocBra is very similar to the optimal ranking. In the second place comes GNCCP, followed by IPFP and then SBPBeam at last.

In Table 3.10, the percentage p-value under 5% is reported for the MUTA-30 data set. The average p-value is 100% for LocBra. Hence, there is a strong correlation between the ranking of LocBra and the optimal ranking. Moreover, GNCCP has scored 70%, higher than SBPBeam and IPFP (both 50%). GNCCP should reject the null hypothesis in 70% of the cases. Regarding the correlation distribution shown in Figure 3.21 chart (a), all the values obtained by LocBra are uniformly in bin 1. This means that LocBra ranking is perfectly correlated with the optimal ranking. GNCCP comes in the second place but the correlation values are distributed in a wide range between [0.2; 1]. IPFP shows poor correlation with the optimal and has negative value (-0.2) for one instance.

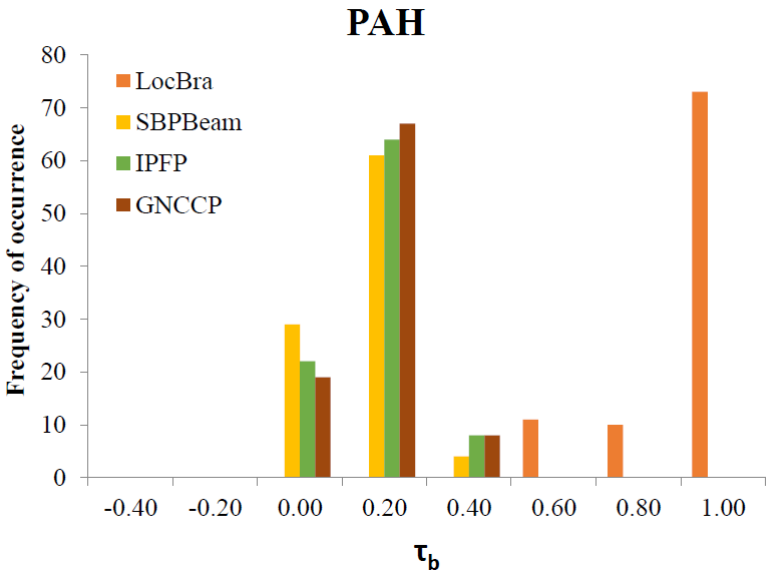
These experiments have empirically demonstrated that LocBra ranking is strongly correlated with the optimal ranking. But more importantly, we could show that the type of heuristics has a strong impact on the the ranking and so on the kNN methods based on GED heuristics.

3.2.3.2 Deadlock 6: Is there a way to specialize a line search method to operate in graph space?

3.2.3.2.1 Motivation As stated earlier, complexity of Problem 11(kNN) depends on $M = |\mathcal{D}|$ and σ the complexity of the dissimilarity function. In a learning-free context, we cannot reduce the factor M (the size of the database). Many researchers decided then to reduce σ thanks to GED heuristics. However, non of the previous work tries to reduce σ considering that the GED solver is involved in a kNN problem. In other words, to the best of our knowledge, all the existing works compared the query graph to each graph in the training set separately without considering the entire problem (GED + kNN problems). When using a kNN method based on a line search, the



(a)



(b)

Figure 3.21: Histograms showing τ_b distribution for each heuristic for MUTA-30 (a) and PAH (b)

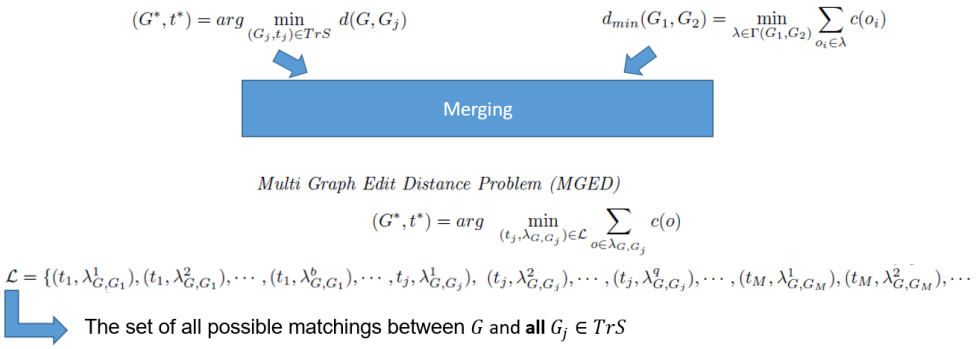


Figure 3.22: Multi graph edit distance in a single picture.

comparison of $G \in TeS$ and each $G_j \in TrS$ is achieved independently. Each distance is treated individually that is the result of a prior comparison cannot help in solving the next comparison. To tackle this issue, we propose a new problem called multi graph edit distance (MGED) where the kNN problem and the GED problem are combined in a single problem. Figure 3.22 expresses the main idea of this new problem. Furthermore, we show that the kNN problem is a special case of the MGED problem.

3.2.3.2.2 MGED problem The MGED can be seen as a merge of the two problems formulated in Problems 11(kNN) and 5(GED). First, let us recall that $\Gamma(G, G_j) = \{\lambda^1_{G,G_j}, \lambda^2_{G,G_j}, \cdots, \lambda^b_{G,G_j}\}$ is the set of all possible matchings between $G \in TeS$ and $G_j \in TrS$. The number of possible matchings b is exponential with respect to the number of vertices in G and G_j.
Now, let $\mathcal{L} = \{(t_1, \Gamma(G, G_1)), \cdots, (t_j, \Gamma(G, G_j)), \cdots, (t_M, \Gamma(G, G_M))\}$ be the set of all possible matchings between G and each graph $G_j \in TrS$ where t_j is the class of the graph G_j. The set $\mathcal{L}$ can be expanded by developing the Γ sets. $\mathcal{L} = \{(t_1, \lambda^1_{G,G_1}), (t_1, \lambda^2_{G,G_1}), \cdots, (t_1, \lambda^b_{G,G_1}), \cdots, t_j, \lambda^1_{G,G_j}),$
$(t_j, \lambda^2_{G,G_j}), \cdots, (t_j, \lambda^q_{G,G_j}), \cdots, (t_M, \lambda^1_{G,G_M}), (t_M, \lambda^2_{G,G_M}), \cdots,$
$(t_M, \lambda^r_{G,G_M})\}$. The MGED problem can be defined as follows:

Problem 12. *Multi Graph Edit Distance Problem (MGED)*

$$(G^*, t^*) = arg \min_{(t_j,\lambda_{G,G_j})\in\mathcal{L}} \sum_{o\in\lambda_{G,G_j}} c(o) \tag{3.34}$$

The former Problem 12(MGED) can be seen as searching the minimum matching (or edit path) among all the matchings between one query graph G and a graph collection. The MGED problem can be extended to find the k minimum matchings by using the *sort* function. The kMGED problem can be defined as follows:

Problem 13. *k-Multi Graph Edit Distance Problem (kMGED)*

$$\mathcal{K} = arg \underset{(t_j,\lambda_{G,G_j})\in\mathcal{L}}{sort} \left(\left(\sum_{o\in\lambda_{G,G_j}} c(o) \right), k \right) \tag{3.35}$$

where *sort* is a function that performs an ascending sort. The kMGED problem expresses the search of the k matchings with the cheapest costs. $\mathcal{K}$ can contain different matchings that belong to a single graph G_j. To respect the kNN problem each matching $\lambda_{G,G_j} \in \mathcal{K}$ should belong to different graphs (λ^1_{G,G_j} and λ^2_{G,G_j} are forbidden). In other words, G_j should appear only once in $\mathcal{K}$. This is modeled by the Constraint 3.36b.

Problem 14. *Constrained k-Multi Graph Edit Distance Problem (CkMGED)*

$$\mathcal{K} = arg \underset{(t_j,\lambda_{G,G_j})\in\mathcal{L}}{sort} \left(\left(\sum_{o\in\lambda_{G,G_j}} c(o) \right), k \right) \tag{3.36a}$$

$$Subject\ to\ \exists! G_j \in \mathcal{K} \quad \forall (t_j, \lambda_{G,G_j}) \in \mathcal{K} \tag{3.36b}$$

where $\mathcal{K}$ is the set of graphs along with their class labels with $(G_j, t_j) \in TrS$. Constraint 3.36b ensures that the pair (G_j, t_j) cannot appear twice. That is, only the best feasible solutions of each GED computation (G and G_j) is selected and thus (G_j, t_j) appears only once. In the worst case, the time complexity of solving the problem of kMGED is exponential in the number of vertices of the graphs G and G_j multiplied by the number of graphs in TrS. In other words, the complexity at the worst case equals to the complexity of the kNN problem $O(M\sigma)$. Apparently, we did not achieve our goal to reduce the kNN complexity thanks to the MGED problem. It is true for the complexity at the worst case. However, experimentally, we can expect that an heuristic algorithm takes advantage of the MGED problem.

3.2.3.2.3 One-tree depth first algorithm to solve the kMGED problem Now, we present a first algorithm to solve the kMGED problem defined in Problem 14(CkMGED). As a final application, the algorithm classifies a query graph G by searching within a single search space.

Algorithm 1 depicts the main steps of the proposed algorithm, called *One-Tree-kMGED*. Lines 1 to 3 correspond to the initialization step. The GED solver is called for the comparison between G_q and G_j (Line 5). The obtained distance d is then added to the list $Dmin$ at the location $k+1$ (line 7). The list of distances is sorted in ascending order while keeping track of IDs (Line 8). In Line 9, the upper bound UB is updated and is given the value of the k^{th} element saved in the distance list $Dmin$ (i.e., $Dmin[k]$). After all the aforementioned steps, the algorithm *One-Tree-kMGED* returns the graphs along with their associated class label $(G_{IDmin[k]}, c_{IDmin[k]})$.

Algorithm 1 One-Tree-kMGED Algorithm

Input: The set TrS: $\{(G_1, t_1), \cdots, (G_M, c_M)\}$, the unknown query graph G_q and the parameter k

Output: The k nearest graphs to G_q from the set TrS with their associated class

1: $Dmin = [+\infty, \cdots, +\infty]$ *A distance of* $k+1$ *elements*
2: $IDmin = [+\infty, \cdots, +\infty]$ *A graph ID list*
3: $UB = +\infty$ *The initial upper bound*
4: **for** $j = 1$ to M **do**
5: d = GED (G_q,G_j,UB)
6: $Dmin[k+1] = d$
7: $IDmin[k+1] = j$
8: $(IDmin, Dmin) = sort_{Dmin}(IDmin, Dmin)$ *sort in an ascending order*
9: $UB = Dmin[k]$
10: **end for**
11: Return $(G_{IDmin[1]}, t_{IDmin[1]}), \cdots, (G_{IDmin[k]}, t_{IDmin[k]})$

3.2.3.2.4 Used GED solver In line 5 of Algorithm 1, any GED solver that takes an upper bound as an input is suitable. We propose to use the depth first algorithm *DF* in [Abu-Aisheh et al., 2015b] (see Section 3.1). It is a branch and bound method and it has the ability to prune the search space thanks to its upper and lower bounds. The solution space is organized as a search

tree. The exploration of the search tree is performed in a depth-first way. Bounding is performed when finding a leaf node with a cost smaller than UB. Pruning is performed by cutting nodes with a larger cost than UB. A node p has a cost $lb(p) = g(p) + h(p)$ where $g(p)$ is the cost of the partial edit path and $h(p)$ is an estimation of the future cost to obtain a complete edit path. If $lb(p) \geq UB$ then the branch is discarded. In order to simply illustrate *One-Tree-kMGED*, Figure 3.23 highlights its idea for $k = 1$. Given a query graph G_q and a learning database TrS, the idea is to consider each search tree S_{qj} of the $GED(G_q,G_j)$ as a sub-tree of the global tree dedicated to the query G_q. The global tree is referred to as T_q. For instance, in Figure 3.23, one can see that the first UB found while exploring the sub-tree S_{q1} of GED(G_q,G_1) is 2. UB is then used as an initial UB of the sub-tree S_{q2} of GED(G_q,G_2) and so on. Such an operation helps in pruning the sub-trees as fast as possible while searching for the nearest neighbor of G_q.

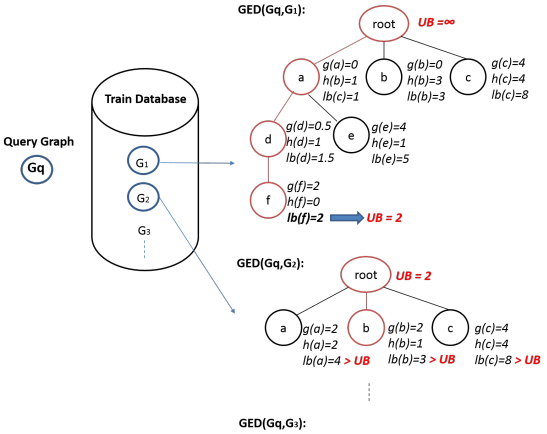


Figure 3.23: *one-Tree-kMGED* when $k = 1$. Given a query graph G_q and graphs in the training set, the problems GED(G_q,G_1), GED(G_q,G_2) and GED(G_q,G_3) are considered as sub-trees of the global tree (T_q). The sub-tree of GED(G_q,G_j) is pruned thanks to UB that is found via GED(G_q,G_1).

3.2.3.2.5 Complexity and time constraint In the worst case, the time complexity of *one-Tree-kMGED* is exponential in the number of vertices of the involved graphs. This case occurs when the first UB does not help in pruning the rest of the search tree and thus all the TrS subtrees need to be explored. One should notice that the complexity in the worst case equals to the complexity of the line search method.

Conceptually speaking, a line search based on *DF* and *One-Tree-kMGED* provide the same neighbors. This statement is true under one assumption that is the time constraint to solve each graph comparison is infinite. Another way to view this assumption is to say that the time limit is never reached by the solvers and the solutions are optimal. On the other hand, *One-Tree-kMGED* does not output the distance of each graph pair (G_q,G_j). The GED computation of each (G_q,G_j) is stopped as soon as the solver proved that no better solutions than the global upper bound UB could be found. In such a case, the UB value is returned as an output of the given graph comparison. The only distances that are guaranteed to be outputted are the GED values of the kNN. The global UB could accelerate the classification time and maybe improve the classification rate.

3.2.3.2.6 Theoretical discussion around the impacts of the parameters As depicted in Algorithm 1, *one-Tree-kMGED* has 3 parameters: the test graph G_q, the training set TrS and

the parameter k for the selection of the nearest neighbors. In this section, a discussion about the impact of *Trs* and k is provided.

3.2.3.2.6.1 Parameter k Regarding *One-Tree-kMGED*, having a big value of k, could have a big impact not only on the classification rate but also on the execution time. The reason is that, the upper bound will be the k^{th} one in the $\{d_{min}\}$ list and not the best upper bound found so far. The k^{th} upper bound is higher than the first upper bound. Consequently, the k^{th} upper bound is likely less capable of cutting the search tree. Such a fact could slow down the algorithm depending on the difficulty of the classification problems.

3.2.3.2.6.2 Ordering the graphs in the training set In Algorithm 1, the graphs in TrS were supposed to be already ordered. However, the question arises: Which order of training graphs should be taken into account in order to prune the search tree as soon as possible? We propose 3 different orderings:

1. RO: TrS is randomly shuffled.
2. CACO: TrS samples are ordered by class.
3. SGPCO: TrS is organized by batches containing one graph of each class.

3.2.3.3 Deadlock 7: What is the impact of GED heuristics in a classification context ?

We aim at answering the question whether the resulting sub-optimal graph edit distances remain sufficiently accurate for classification tasks. On its own, a kNN classifier does not aim at maximizing a classification rate. A kNN classifier relies on a dissimilarity function that drives its objective. In our context, the dissimilarity function is a GED method. All the GED methods aim at minimizing the sum of the edit operations costs. The cost is the piece of information that link the GED problem to the classification problem. From this statement, two sub questions can arise. 1) Are heuristics sufficiently accurate for classification tasks when cost functions are chosen based on expert knowledge? 2) Are heuristics sufficiently accurate for classification tasks when cost functions are learned to maximize a classification rate?

To answer these questions, classification experiments must be performed. Heuristics should be compared with exact methods.

3.2.3.3.1 1) Are heuristics sufficiently accurate for classification tasks when cost functions are chosen based on expert knowledge? This paragraph is organized as follows: First the data sets are presented. Then the way of computing the cost functions is described. Classification rate are summarized and an analysis is given.

Table 3.11 synthesizes the characteristics of each of the selected data sets in terms of the number of graphs in both train and test sets and the number of classes. The data sets are selected because the computation of optimal solutions is possible.

The cost functions $c(\cdot)$ for each data set are summed up in Table 3.12. These cost functions are not chosen randomly. Solving the GED with these cost functions correspond to solve the Maximum Common Subgraph (Problem 2).

The selected classifier is a 1NN classifier which has the advantage of being parameter free. Classification results are presented in Table 3.13. JH is an exact method. All optimal solutions were computed by JH. BP and BS-10 are two heuristics. *Acc* is the classification rate on the test set. *time* is the average time in milliseconds to classify a query. From this table, we can make some

Database	# Train	# Test	Comment
PAH	22	22	2 classes
MAO	16	16	2 classes
LETTER-HIGH	750	16	10 classes

Table 3.11: Data sets selected from the ICPR'2016 contest.

	Node Cost function	Edge Cost function	Node/Edge substitution	Node/Edge Insertion	Node/Edge Deletion
PAH	Kronecker delta	Kronecker delta	0	3	3
MAO	Kronecker delta	Kronecker delta	0 or 3	3	3
LETTER-HIGH	Euclidean distance	Kronecker delta	$L2$ norm for nodes. Edge are without attributes.	1	1

Table 3.12: Learning-free cost functions.

comments. Classification rates are not negatively impacted by heuristics. Heuristics can obtain higher classification rates than exact methods.

Now, we can try to analyze this behaviour.

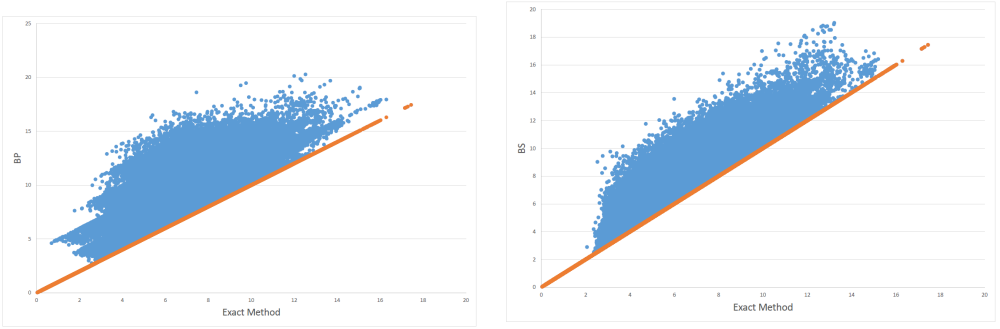
3.2.3.3.2 Why heuristics can provide good classification results? As mentioned before, the distances found by heuristic methods are equal to, or larger than, the optimal distances. Such distances are feasible solutions and represent upper bounds of the GED problem. The correlation between optimal and suboptimal methods can be seen in Figure 3.24. These scatter plots give a visual representation of the accuracy of the two heuristics BS-10 (BeamSearch) and BP (LSAP-based) on the LETTER-HIGH data set. The optimal solution were computed by the method called JH. Based on the scatter plots given in Figure 3.24, we find that BS approximates small distance values accurately, i.e. all small suboptimal distances are equal to the optimal distances. On the other hand, large distance values are overestimated quite strongly. Based on the fact that graphs within the same class usually have a smaller distance than graphs belonging to two different classes. This means that the suboptimality of BS mainly increases interclass distances, while intra-class distances are not strongly affected. A similar conclusion can be drawn for BP. Many of the small distance values are not overestimated, while higher distance values are increased due to the suboptimal nature of the approach.

Method	PAH		MAO		LETTER-High	
	Acc	time (ms)	Acc	time (ms)	Acc	time (ms)
JH	0.6363	31765.5	0.8125	11009	0.8346	534.66
BP	0.6363	34.81	0.75	26.75	0.836	80.63
BS-10	0.6818	2928.04	0.8125	57.68	0.8346	227.44

Table 3.13: Classification rate obtained by exact and heuristic methods.

Moreover, for a nearest-neighbor classifier, small distances have more influence on the decision than large distances. Hence no serious deterioration of the classification accuracy occurs when heuristic algorithms are used instead of an exact method.

We can conclude that on the tested data sets, the classification accuracy of the 1NN classifier is not negatively affected by using the suboptimal distances. This is due to the fact that most of the overestimated distances belong to inter-class pairs of graphs, while intra-class distances are not strongly affected. Obviously, intraclass distances are of much higher importance for a distance based classifier than inter-class distances. In other words, through the approximation of the edit distances, the graphs are rearranged with respect to each other such that a better classification becomes possible. Graphs which belong to the same class (according to the ground truth) often remain near, while graphs from different classes are pulled apart from each other. Obviously, if the approximation is too inaccurate, the similarity measure and the underlying classifier will be unfavorably disturbed. Probably, the concerned data sets are not difficult enough to put forward the impact of heuristics in a classification context. A distance of zero is likely to appear and these distances are well computed by heuristics. Extending the number of k nearest neighbors could be a way to see the differences between exact and heuristic methods. The computation of distances between farther neighbors could be more challenging for heuristics.



BP(left) and BS-10(right) on the LETTER dataset.

Figure 3.24: Scatter plots of the optimal edit distances (x-axis) and the suboptimal edit distances (y-axis). Orange dots are optimal distances.

3.2.3.3.3 Question 2: Are heuristics sufficiently accurate for classification tasks when cost functions are learned? First the data set and the cost functions are presented. Then the way of computing the cost functions (c) is described. Classification rate are summarized and an analysis is given.

Table 3.14 synthesizes the characteristics of each of the selected data sets in terms of the number of graphs in both train and test sets, the average and maximum number of vertices and edges and the attributes on both of them.

Each data set has specific edit cost functions that define how the insertion, deletion and substitution are achieved [Riesen and Bunke, 2010b]. In most of the datasets, two non-negative meta parameters are associated: ($\tau_{vertex} \in \mathbb{R}$ and $\tau_{edge} \in \mathbb{R}$) where τ_{vertex} denotes a vertex deletion or insertion costs whereas τ_{edge} denotes an edge deletion or insertion costs. A third meta parameter $\alpha \in [0, 1]$ is integrated to control whether the edit operation cost on the vertices or on the edges is more important. Table 3.15 reports the cost functions of each of the included data sets as well

Table 3.14: The characteristics of the datasets included in the experiments.

Dataset	GREC	Protein	Muta	Fingerprint	Webpage	House-Hotel
#train graphs	286	200	1500	378	780	51
#valid graphs	286	200	500	3OO	780	20
#test graphs	528	200	2337	1532	780	70
$\overline{vertices}$	11.5	32.6	30.3	5.38	186.04	30
$\overline{edges}$	12.2	30.8	79	8.8	104.03	62.1
Max vertices	25	40	71	26	785	30
Max edges	30	149	112	48	524	79
Vertex labels	x,y coordinates	Type and amino acid sequences	Chemical symbol	None	Word's frequency	60 size feature
Edge labels	Line type	Type and length	Valence	Orientation	Section label	Distance

as their meta parameters. Each database is divided into three disjoint subsets, viz. the training, the validation, and the test set. The elements of the training set are used as prototypes in the 1NN classifier. The validation set is used to determine the values of the meta parameters τ_{node}, τ_{edge} and α that maximizes a classification rate for the method called BP. The cost function is parametrized with τ_{node}, τ_{edge} and α (.i.e. $c(i,k;\tau_{node},\tau_{edge},\alpha)$). This cost function is optimized only for the BP heuristic. Is the learned cost function useful for other heuristics? This is open question. This general question can be refined if a link does exist between methods. For example, if BP is an upper bound of the method called DF then does the optimized cost function $c^*(.)$ maximizes a classification rate for DF?

Table 3.15: The cost functions and meta parameters of the datasets.

Dataset	GREC	Muta	Protein	Fingerprint	Webpage	House-Hotel
τ_{vertex}	90	11	11	0.7	2	3
τ_{edge}	15	1.1	1	0.5	2	3
α	0.5	0.25	0.75	0.75	0.5	0.5
Vertex substitution function	Extended euclidean distance	Dirac function	Extended string edit distance	Absolute value	Dirac function	Dirac function
Edge substitution function	Dirac function	Dirac function	Dirac function	Absolute value	Absolute Value	Dirac function
Reference of cost functions	Riesen and Bunke [2010b]	Riesen and Bunke [2010b]	Riesen and Bunke [2010b]	Riesen and Bunke [2010b]	Riesen and Bunke [2010b]	Moreno-García et al. [2016]

In Table 3.16, the results achieved on all the datasets are presented. Note that the computation time corresponds to the average time needed per dissimilarity in milliseconds (ms).

The results show that on all the datasets, *one-Tree-kMGED* was always faster than the classical *DF* approach. It also improved the classification rate of *DF* on both Protein and Muta. *one-Tree-kMGED* could improve *UB* while moving from one comparison to another. As a consequence, it pruned unfruitful parts of the global search tree and found smaller distances. As mentioned in the previous Paragraph 3.2.3.3.3, a better minimization of the GED or the MGED problems does not always lead to a higher classification rate even when the cost function is optimized to improve a

Table 3.16: Classification results on five datasets where *time* refers to the average time in milliseconds needed by each dissimilarity computation whereas *Acc* refers to the classification accuracy. The best results are marked in bold style. Note that k was fixed to 1.

	GREC		Protein		Muta		Fingerprint		WebPage		House-Hotel	
	time	Acc	time	Acc	time	Acc	time	Acc	time	Acc	time	Acc
one-Tree-k-DF (CAFO)	136.31	**98.5**	320.77	47	70.56	**72.41**	30.25	61.68	152.48	**21**	395.66	**98,57**
one-Tree-k-DF (SGPCO)	**51.01**	**98.5**	306.48	47	**69.23**	71.05	29.06	61.68	**48.91**	**21**	370.43	**98,57**
one-Tree-k-DF (RO)	54.13	**98.5**	291.60	47	69.45	71.28	**28.76**	61.68	51.61	**21**	363.37	**98,57**
DF	491.87	**98.5**	426.90	42	487.43	70	168.45	**63.83**	470.03	**21**	485.05	**98,57**
BS-1	242.08	**98.5**	**127.35**	42.5	434.61	55.5	74.35	62.46	426.22	12.4	**108.18**	**98.57**
BS-100	293.45	58.7	475.69	31.0	486.71	55.5	211.74	10.3	499.21	4.3	478.91	**98,57**
BP	217.81	**98.5**	295.20	**52**	352.36	70	42.60	60.40	466.81	**21**	308.71	**98.57**
FBP	97.63	**98.5**	197.12	38.5	250.57	70	36.53	61.35	449.06	15	189.07	**98.57**

classification rate.

When comparing *one-Tree-kMGED* to *BP*, one can see that *one-Tree-kMGED* was 4.2 times faster (on GREC), 3.6 times faster (on Muta) and 9.5 times faster (on WebPage). On the other hand, on Fingerprint and House-Hotel, the speed results of the two methods were quite similar. This is due to the small number of graphs in these datasets and thus the advantage of using prior UB in *one-Tree-kMGED* cannot be fully revealed. Moreover, on House-Hotel, *BS-1* was the fastest. House-Hotel has easy graphs to classify (with only 2 classes) and that is why all the methods obtained 98.5% as a classification rate. Another interesting remark is that *one-Tree-kMGED* succeeded in improving the classification rate on Muta. However, on Protein, it was less accurate than *BP*. Despite the fact that *BS-1* was faster than *one-Tree-kMGED* on Protein, the accuracy of *BS-1* was lower. As a general conclusion, *one-Tree-kMGED* was the fastest algorithm, except on Protein and CMU where *FBP* won. This point is explained by the fact that the number of train graphs in both of them was quite small so that the interest of merging all sub-problems into a unique one is not useful, see Table 3.14.

The results of Table 3.16 confirms that a cost function originally optimized for the method called *BP* can be applied to different heuristics. Results obtained by *BP* can also be improved thanks to an effective GED solver or a MGED algorithm.

3.2.4 Summary

During the thesis of Mostafa Darwiche, the ranking experiment showed that the type of heuristics has a great impact on the order of the graphs returned by a kNN method [Darwiche et al., 2018, in pressb]. During the PhD of Zeina Abu-aisheh, a new problem referred to as multi graph edit distance (MGED) was defined. The MGED can be seen as searching the k minimum matchings (or edit paths) among all the matchings between one query graph G and a graph collection $\mathcal{D}$. Under some constraints, the problem of finding kNN falls within the MGED problem [Abu-Aisheh et al., 2017a]. A classification task was led to evaluate GED solvers and the MGED solver. In this test, the cost function is of first interest. It has been showed that cost functions, optimized for a given GED method, could be transferred with success to different GED methods.

Finally, a comparison between exact and heuristic methods in a classification context has shown that the classification accuracy of the 1NN classifier is not negatively affected by suboptimal distances. However, it cannot be generalized unconditionally since if the approximation is too inaccurate, the similarity measure and the underlying classifier will be unfavorably disturbed. In the same vein, we must take caution because conclusions that are drawn for 1NN might be different if the number of neighbors gets larger. The computation of distances between farther neighbors could be more challenging for heuristics.

Chapter 4

Structural machine learning for graph matching and graph classification

Contents

4.1 Graph matching . . . 80
4.1.1 State of the art of learning graph matching . . . 80
4.1.1.1 Shallow methods . . . 82
4.1.1.2 Deep methods . . . 85
4.1.1.3 Summary . . . 88
4.1.2 Open problems . . . 92
4.1.3 Contributions . . . 93
4.1.3.1 Deadlock 8 : How to deal at a fine level with insertion and deletion costs? . . . 93
4.1.3.1.1 Motivation . . . 93
4.1.3.1.2 Details . . . 93
4.1.3.2 Deadlock 9: Can an heuristic output solutions closer to optimality thanks to machine learning? . . . 95
4.1.3.2.1 Motivation . . . 95
4.1.3.2.2 Details . . . 95
4.1.4 Summary . . . 100
4.2 Graph classification . . . 101
4.2.1 State of the art from a machine learning viewpoint . . . 101
4.2.1.1 Learning graph prototypes . . . 103
4.2.1.1.1 Set prototypes . . . 104
4.2.1.1.2 Generalized prototypes . . . 105
4.2.1.2 Learning graph (dis)similarity measure . . . 106
4.2.2 Open problems . . . 106
4.2.3 Contributions . . . 107
4.2.3.1 Deadlock 10: Learning graph distance for classification with local parameters for nodes and edges . . . 107
4.2.3.1.1 Motivation . . . 107
4.2.3.1.2 Details . . . 108

4.2.3.2 Deadlock 11: Learning graph matching and graph prototypes in a hierarchical manner . . . 111
4.2.3.2.1 Motivation . . . 111
4.2.3.2.2 Details . . . 113
4.2.4 Summary . . . 118

This section is split into two parts graph matching (Section 4.1) and graph classification (Section 4.2). Each part is then broken down in three steps. First, the state of the art is summed up. Second, deadlocks and open problems are expressed. Finally, contributions are presented. The section is focused on machine learning techniques. An introduction that presents the foundations of the learning theory is presented in Appendix B. References about our contributions are given in the summary sections (Section 4.1.4 and Section 4.2.4). Note that the state of the arts do not include our work. This is intentionally done to highlight how our contributions help to release the deadlocks.

4.1 Graph matching

4.1.1 State of the art of learning graph matching

The learning graph matching problem can be framed as the minimization of an error rate on the number of correctly matched graph components. This can be seen as minimizing the average Hamming distance between ground-truth's correspondences and the computer generated correspondences on a data set of graph pairs. The training set is then defined as $TrS = \{((G_1, G_2)_k, y_k^{gt})\}_{k=1}^{M}$. (G_1, G_2) is a graph pair and y^{gt} is the ground-truth's correspondences. A computer generated matching that is considered as a prediction is written y from the machine learning viewpoint. A graph matching solution is defined as a subset of possible correspondences $y \subset \mathcal{V}_1 \times \mathcal{V}_2$ or $y \subset V_1 \times V_2$ depending on the graph matching problem. Correspondences are represented by a binary assignment matrix $Y \in \{0,1\}^{n_1 \times n_2}$, where n_1, n_2 denote the sizes of the vertex sets. We denote by $y \in \{0,1\}^{n_1 \cdot n_2}$, a column-wise vectorized replica of Y. Generally speaking, the learning problem can be stated as follows:

Problem 15. *Learning graph matching problem (LGM)*

$$\min_{W} \sum_{((G_1,G_2),y^{gt}) \in TrS} \min_{y \in \Gamma(G_1,G_2)} l(y, y^{gt}, W) \tag{4.1}$$

Where Γ is the set of all possible matchings between G_1 and G_2. l is the loss function and a possible choice is the Hamming distance $l = \|y^{gt} - y(W)\|_1$. W are invariants (trainable parameters) over the data set TrS and y are variables of the graph matching problem.

Inside this problem, many learning problems arise. Node/edge attributes can be learned. This procedure is assimilated to a feature extraction step. Another area where learning algorithms can be applied is the learning of node/edge (di)ssimilarity function. Finally, the graph matching algorithm itself can be improved or replaced by a machine learning algorithm. In Figure 4.1, an overview of these three concepts is provided. The literature about graph matching is decomposed into two parts: shallow and deep methods. Deep methods refers to models composed of a hierarchy of sub-models (representations). "Deep" architectures take advantages of properties of the data such as compositionality. That is data can be represented by a composition of (simple) models. There are many different methods for learning graph matching but the general learning scheme is depicted in Figure 4.2. The ML method takes a graph pair as an input and output matching that depends on parameters W. The predicted matching is compared to the ground-truth and errors are given back to the ML method to adapt its parameters.

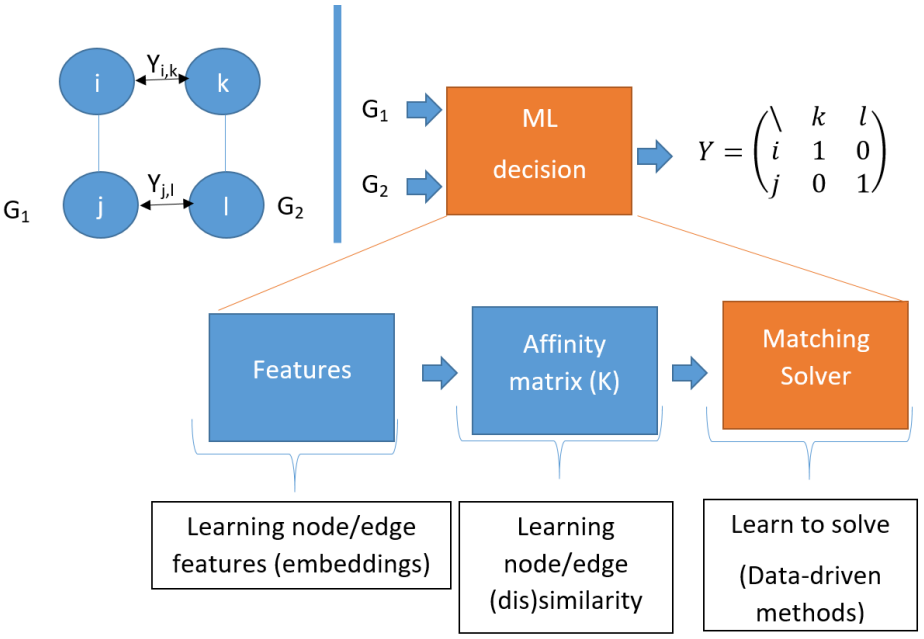


Figure 4.1: An overview of the learning graph matching problem. Three sub-problems arise as learning features, learning dissimilarity and learning to match.

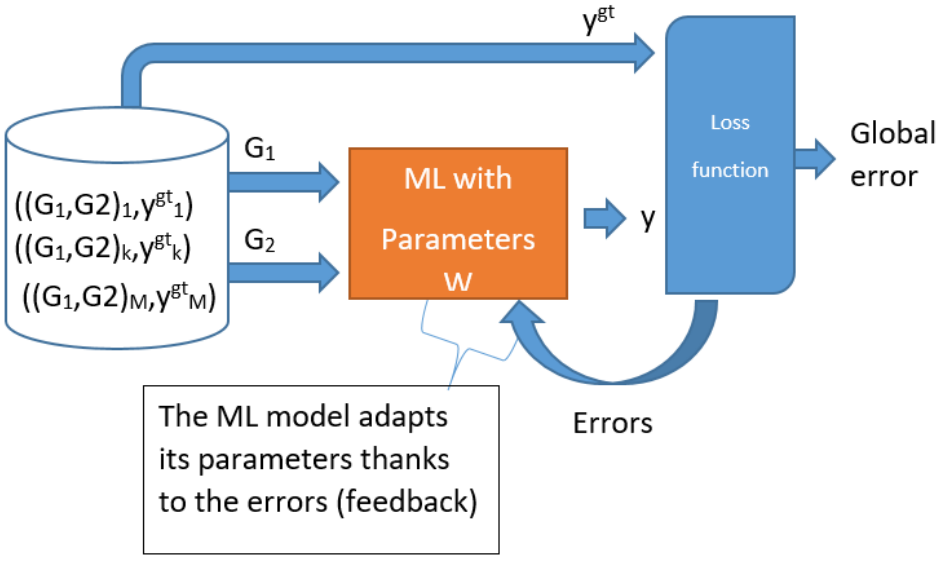


Figure 4.2: A general framework for supervised learning of graph matching.

In addition, we only focus on techniques producing a matching at the end. Techniques such as metric learning methods that do not cope with this constraint are not detailed [Riba et al., 2018].

The papers in the literature mainly differ on three aspects :

- How the Problem15 (LGM) is minimized.
- How the losses are employed.
- How the parameters are introduced.
- How the node/edge insertions and deletions are considered.
- Does the method involves a graph matching solver or not ?

Now, we present a deeper insight of the different methods.

4.1.1.1 Shallow methods

The shallow methods focuses on learning the cost functions ($c(.)$)). To better characterize the state of the art, we introduce two notations $d_V(.)$ and $d_E(.)$ that are derived from the cost function $c(.)$. $d_V(.)$ and $d_E(.)$ denote specific cost functions between vertices and edges, respectively. With this consideration, the objective function defined in Equation 3.2 of the Model GMIQP can be rewritten as follows :

$$d(G_1, G_2, y) = \sum_{y_{ik}=1} d_V(i,k) + \sum_{y_{ik}=1} \sum_{y_{jl}=1} d_E(ij,kl) \tag{4.2}$$

Equation 4.2 must be understood as the sum of vertex and edge dissimilarities when y variables equal to 1. Equation 4.2 must be minimized.

In this the same way of splitting cost functions for nodes and edges, the matching problem based on similarity ($s_V(.)$ and $s_E(.)$) defined in the Model SGMIQP can be rewritten as follows :

$$s(G_1, G_2, y) = \sum_{y_{ik}=1} s_V(i,k) + \sum_{y_{ik}=1} \sum_{y_{jl}=1} s_E(ij,kl) \tag{4.3}$$

Equation 4.3 must be maximized.

The first series of papers are learning methods dedicated to the problem expressed by the Model SGMIQP. It means that only substitutions are concerned.

The method of Caetano et al. [Caetano et al., 2009] aims at learning global features to facilitate the matching. Parameters to be learned are located on the node and edge features as shown in Equation 4.4. Parameters w_E and w_V are vectors of real values whose sizes are equal to the size of the vertex or edge features.

$$s(G_1, G_2, y, w_V, w_E) = \sum_{y_{ik}=1} s_V(i,k,w_V) + \sum_{y_{ik}=1} \sum_{y_{jl}=1} s_E(ij,kl,w_E) \tag{4.4}$$

They use graphs with numeric attributes so s_v is a 60-dimensional node similarity function for appearance similarity and s_v is a simple binary edge similarity for edges. The learning procedure is iterative and aims at maximizing the number of correctly matched graph components between the ground-truth y^{gt} and the estimated matching y. At each iteration the graph matching problem is solved by a *branch and price* method and the parameters $w = [w_V, w_E]$ are updated by the gradient descent method. Weights are global over the whole data set. The results reveal that learning can substantially improve the performance of standard graph matching algorithms. In particular, they found that **simple linear assignment** with such a **learning** scheme outperforms the **Graduated Assignment method**.

Leordeanu et al. [Leordeanu and Hebert, 2009, Leordeanu et al., 2012] used the same strategy than Caetano et al. [Caetano et al., 2009]. They parametrized functions s_V and s_E with weight vectors. They showed for the first time how to perform parameter learning in an unsupervised fashion, that is when no correct correspondences between graphs are given during training. The assumption is that the expected values of the second order scores $s(ij, kl)$ do not depend on the particular assignments ij or kl, but only on whether these assignments are correct or not. In average over many matchings, the pairs of correct assignments are expected to agree in appearance and so $\mathbb{E}[s(ij, kl)]$ should be high. The leading eigenvector ($v \in \mathbb{R}^{n1.n2}$) of the affinity matrix $K \in \mathbb{R}^{n1.n2 \times n1.n2}$ (Model SGMIQP) should follow the same principle. In average over many matchings, all correct assignments ik must have high values v_{ik} and wrong assignments (jl) should have low value v_{jl}. In other words, authors assume that eigenvector values are higher on average for correct assignments than for wrong ones. Then the objective is to maximize the correlation between v and its binary version $b(v)$ (that is, the binary solution returned by the matching algorithm): $\max_W \sum_{i=1}^{M} transpose(v_W^{(i)}) b(v_W^{(i)})$. Note that the construction of K depends on parameters W through s_V and s_E functions so the leading eigenvector v depends on W too. So the notation v_W appears.

They showed empirically that unsupervised learning is comparable in efficiency and quality with the supervised one, while avoiding the tedious manual labeling of ground truth correspondences. They also verified experimentally that the unsupervised learning method can improve the performance of several state-of-the art graph matching algorithms. This statement was observed on graphs built from images (CMU-House/ Hotel, VOC-Cars/Motorbikes).

In [Leordeanu et al., 2009], the paper focused on a Frank-Wolfe like method called IPFP but also a learning procedure is activated. They used the supervised version of the graph matching learning method from [Leordeanu and Hebert, 2009].

On specific data sets, learning was effective, improving the performance by more than 15% on average, for all learning-free methods.

The work of Torresani et al. [Torresani et al., 2008] formulated the matching task as an energy minimization problem by defining a complex objective function of the appearance and the spatial arrangement of the features. The objective function is parametrized by 4 weights to be learned. This approach can be viewed as adopting 4-dimensional s_V ($s_V : \mathbb{R}^4 \times \mathbb{R}^4 \to \mathbb{R}$) and s_E functions for measuring appearance dissimilarity, geometric compatibility, and occlusion likelihood. As presented in Section 3.1, the paper [Torresani et al., 2008] is mainly focused on solving the graph matching problem formulated as a CRF problem. The method is called *DD* and is based on dual decomposition. However, they used the method of Liu et al. [Liu et al., 2005] to learn the parameter values for the graph matching model from examples. They applied Nonlinear Inverse Optimization [Liu et al., 2005] (NIO) to learn non-negative parameters. They used *DD* within NIO to optimize the learning objective.

In the next bunch of papers, the learning techniques are dedicated to the graph edit distance and so a special attention is given to deletion and insertion costs. In [Cortés and Serratosa, 2015], a method for learning the real numbers for the insertion $c(\epsilon \to k)$ and deletion $c(i \to \epsilon)$ costs on nodes and edges is proposed. An extension to substitution costs is presented in [Cortés and Serratosa, 2016]. The training set is composed of M observations. Each observation is composed of a pair of graphs $(G_1, G_2)_i$ and also the ground truth correspondence y^{gt}. The computer-generated correspondence y depends on the costs $K_V \in \mathbb{R}$ and $K_E \in \mathbb{R}$. $K_V = d_V(\epsilon, k) = d_V(i, \epsilon)$ and $K_E = d_E(\epsilon, kl) = d_E(ij, \epsilon)$ are scalar values to be learned. The loss function is expressed as

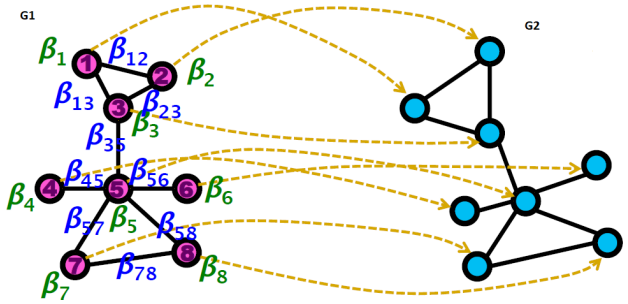


Figure 4.3: Local parametrization of graph matching.

follows:

$$l(K_V, K_E) = \frac{1}{M} \sum_{(G_1,G_2),y^{gt} \in TrS} l_0(G_1, G_2, y^{gt}, y, K_V, K_E) + \lambda \Omega(K_V, K_E) \quad (4.5)$$

$$l_0(G_1, G_2, y^{gt}, y, K_V, K_E) = \left(d(G_1, G_2, y, K_V, K_E) - d(G_1, G_2, y^{gt}, K_V, K_E)\right)^2 \quad (4.6)$$

$$\Omega(K_V, K_E) = || \, [K_V, K_E] \, ||_2^2 \quad (4.7)$$

The loss function l is composed of a data-oriented function l_0 and a regularization term. Please see Appendix B for more details about these terms. l_0 gauges how far the computer generated matching is from the ground-truth matching. Clearly, using this loss function, authors assume that two correspondences that are close to each other (small Hamming distance) tend to achieve similar costs. Although this relation is not true for all graphs and correspondences, the empirical evaluation tends to show that it is true for most of the graphs and correspondences in the considered datasets. Ω is the quadratic regularization term based on the inner product of the weights to be optimized in order to prevent overfitting and underfitting (see Appendix B). The loss function l is minimized by the Nelder–Mead method also called downhill simplex method [Nelder and Mead, 1965] that is a non-linear optimization heuristic.

All the previous works aimed at learning common weights shared among all the edges' and nodes. Weights are global over a graph pair and not local to a given pair of nodes or edges. In other words with an analogy to probability, it could be more appealing to compute the conditional probability $Pr(y^{gt}|node, parameters)$ rather than $Pr(y^{gt}|parameters)$

To overcome this problem, the discriminative weight formulation was introduced by [Cho et al., 2013] and it can assign different parameters for individual node and edge matching as follows :

$$s(G_1, G_2, y, \beta) = \sum_{y_{ik}=1} \beta_i . s_V(i, k) + \sum_{y_{ik}=1} \sum_{y_{jl}=1} \beta_{ij} . s_E(ij, kl) \quad (4.8)$$

β is in this case a vector of parameters indexed by the edge/node matching. Equation 4.8 can be used to parametrize a graph matching problem. The Figure 4.3 depicts this mechanism. The learning problem is to learn the parameters to fit a given objective specified by the loss function. The learning problem turns into the minimization of the Hamming distance measuring the quality of a predicted matching y against its ground truth y^{gt}. The Hamming distance $\|y^{gt} - y\|_1$ is not continuous anywhere and makes the problem not convex. To leverage this difficulty, the normalized Hamming distance is favored and it can be rewritten as follows:

$$l(y, y^{gt}) = 1 - \frac{1}{\|y\|_F^2} y . y^{gt} \quad (4.9)$$

Finally, the learning problem is minimized by a constraint quadratic programming method [Joachims et al., 2009]. The discriminative weight formulation can be seen as a generalization of previous methods [Caetano et al., 2009, Torresani et al., 2008]. The paper [Cho et al., 2013] goes beyond the graph matching problem because the authors proposed also a way to learn/generate the graph G_1. The features and the structure of G_1 are involved in the optimization procedure. The similarity function is the dot product of two attributes:

$$s_V(i,k) = s_V(\mu_1(i), \mu_2(k)) = s_V(a_i, a_k) = a_i^T.a_k$$

Where a_i is a numeric feature vector of node i. Then, the attributes (a_i) of the graph G_1 can be factored out and combined with the weights to be learned:

$$\begin{aligned} s(G_1^*, G_2, y, \beta) &= \sum_{y_{ik}=1} (\beta_i \odot a_i)^T.a_k + \sum_{y_{ik}=1} \sum_{y_{jl}=1} (\beta_{ij} \odot a_{ij})^T.a_{kl} \\ s(G_2, y, W) &= \sum_{y_{ik}=1} (W_i)^T.a_k + \sum_{y_{ik}=1} \sum_{y_{jl}=1} (W_{ij})^T.a_{kl} \end{aligned} \tag{4.10}$$

$\odot$ is an element-wise product. The parameters to be learned are β_i, a_i, β_{ij} and a_{ij}. They are combined in the matrix W to cope with the multidimensional nature of graph attributes. The graph G_1^* is a fully-connected graph where node and edge features are learned.

In [Riesen and Ferrer, 2016], a completely different framework was designed. The main contribution is the prediction of whether two nodes match or not thanks to conventional machine learning tools. The node assignment is represented by a vector of 24 features. These numerical features are extracted from the node-to-node cost matrix $\mathbf{C}$ which was used for the original matching process (called BP algorithm). Then, using the assignments derived from exact graph edit distance computation as the ground truth, each computed node assignment is labeled correct or incorrect. This set of labeled assignments is used to train an SVM endowed with a Gaussian kernel to classify the assignments computed by the approximation as correct or incorrect. This method does not require solving a graph matching problem to predict pairwise node assignment however no guarantee is given to output a feasible solution. The prediction does not rely on a graph matching method.

4.1.1.2 Deep methods

A part from the graph matching community, A. Nowak et al in [Nowak et al., 2017, Nowak et al., 2017] presented a note on learning algorithms for the QAP. They studied data driven approximations to solve it. Especially, since the QAP can be modelled by a graph, they focused on a Graph Neural Network (GNN) model [Scarselli et al., 2009]. An introduction and a review about GNN methods are presented in Appendix C. A graph is processed by a set of units. Each unit corresponds to a node of the graph. Units are linked according to the input graph connectivity. This neural network alternates between applying linear combinations of local graph operators, such as the graph Laplacian, and node-wise non-linearities, and has the ability to model some forms of non-linear message passing from a layer to another. This GNN model can answer both graph regression [1] $f(G) \in \mathbb{R}^d$ and node regression problems $f(G,n) \in \mathbb{R}^d$. $f(G,n)$ maps a graph and one of its nodes into an d-dimensional Euclidean space. An application to subgraph isomorhpism is prensented in [Scarselli et al., 2009]. Given a subgraph S in a larger graph G, the function $f(G,n)$ that has to be learned is such that $f(G_i, n_{i,j}) = 1$ if the node $n_{i,j}$ belongs to a subgraph of G_i, which is isomorphic to S, and, $f(G_i, n_{i,j}) = -1$ otherwise. The presented results are not good and cannot be compared with those achievable by other specific methods for subgraph isomorhpism, which are faster and more accurate. In [Nowak et al., 2017], a simpler GNN is presented. Note

[1] Note that in most regression problems, the mapping is to a vector of reals while in classification problems, the mapping is to a vector of integers. Here, for simplicity of exposition, we will denote only the regression case. See Appendix B for more details about classification and regression. See Appendix C for node classification with GNN.

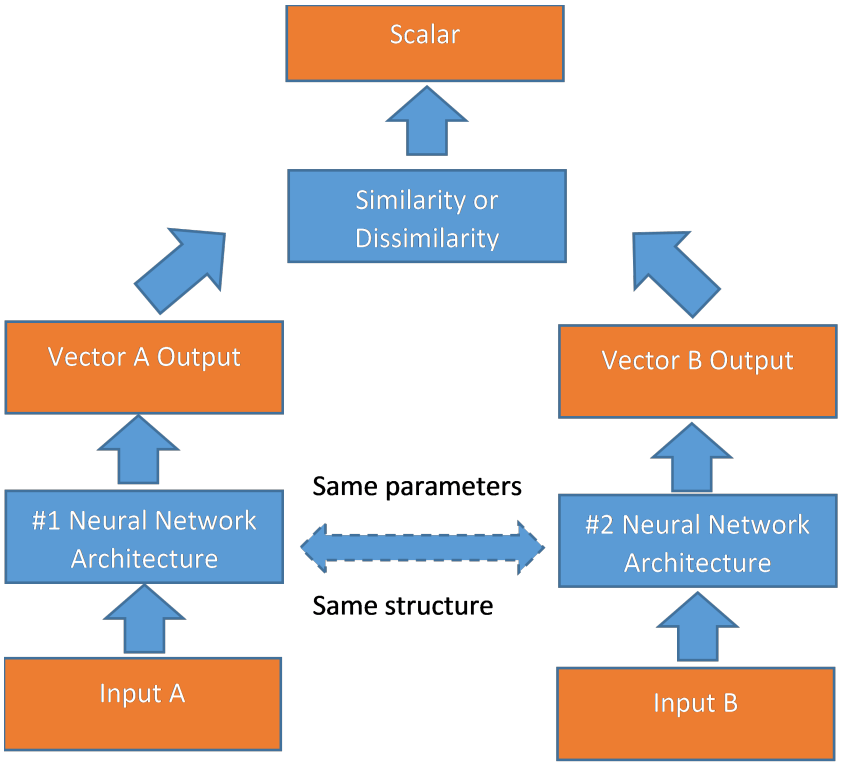


Figure 4.4: Siamese architecture: a structural definition.

that this architecture is detailed in Appendix C. They consider this GNN and train it to solve random instances of the QAP. Given a pair of graphs G_1, G_2 with N nodes each, they consider a siamese GNN encoder producing normalized embeddings E_1, $E_2 \in \mathbb{R}^{N\times d}$. Siamese neural network is a class of neural network architectures that contain two or more identical networks. Identical here means they have the same configuration with the same parameters and weights. Parameter updating is mirrored across both networks (see Figure 4.4 for a structural definition).

Clearly, the siamese GNN produces node embeddings. Those embeddings are used to predict a matching as follows. They first computed the outer product $Z = E_1 E_2^T$, that they then mapped to a stochastic matrix by taking the softmax along each row/column (Sinkhorn-Knopp algorithm [2]). Let us assume that the results is a bi-stochastic matrix $Z \in \mathbb{R}^{N\times N}$ represents node-to-node matching similarity. Finally, they used standard cross-entropy loss to predict the corresponding permutation index. The loss function can be written as follows :

$$l = -\sum_{(G_1,G_2),y^{gt}\in TrS} \sum_{i=0}^{|G_1|}\sum_{k=0}^{|G_2|} Y_{i,k}^{gt} \log Z_{i,k}$$

where Y^{gt} is the ground-truth matching for the graph pair G_1, G_2. Figure 4.5 depicts the overall architecture.

To build the data set TrS, they considered G_1 to be a random Erdos-Renyi graph. The graph G_2 is a small perturbation of G_1 so the matching can be trivially deduced as the Identity matrix $I \in \{0,1\}^{N\times N}$. All graphs have 50 nodes and the density is around 0.2. The GNN method is out

[2] A simple iterative method to approach the double stochastic matrix is to alternately rescale all rows and all columns of $E_1.E_2^T$ to sum to 1

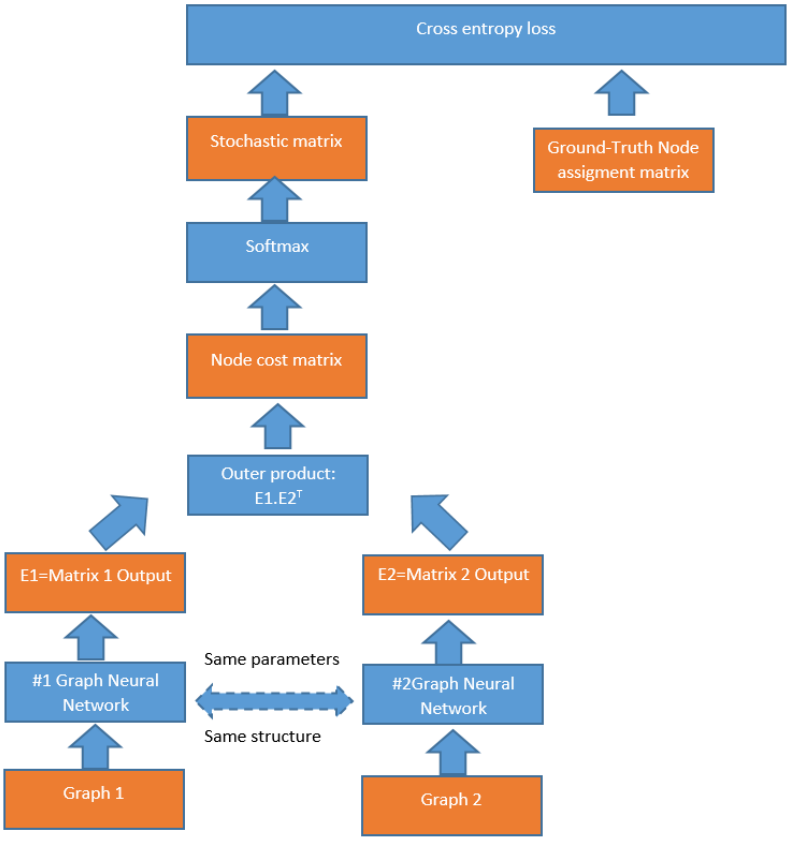


Figure 4.5: Siamese architecture for graph matching.

performed on these graphs by classical QAP solvers. Similarly, another experiment is performed on regular graphs with more symmetric structures. On these regular graphs, the GNN outperformed classical QAP solvers. The source code is available[3].

Recently, [Zanfir and Sminchisescu, 2018] present an end-to-end model that makes it possible to learn all parameters of the graph matching process, including the node and edge features, represented as deep feature extraction hierarchies. The challenge is in the formulation of the different matrix computation layers of the model in a way that enables the consistent, efficient propagation of gradients in the complete pipeline from the loss function, through the combinatorial optimization layer solving the matching problem, and the feature extraction hierarchy. The $\mathcal{NP}$-hard graph matching problem is relaxed by dropping both the binary and the mapping constraints. The model to be solved is then :

Model 6. *L2-norm relaxed QAP (RSGMIQP)*

$$y^* = \underset{y}{\text{argmax}} \quad y^T K y \tag{4.11a}$$

$$\textit{subject to} \quad y \in [0,1]^{|V_1|\cdot|V_2|} \tag{4.11b}$$

$$\|y\|_2 = 1 \tag{4.11c}$$

The optimal y^* is then given by the leading eigenvector of the matrix K. The main components of the approach are shown in Figure 4.6. One may notice that the inputs are two images and two graphs of keypoints. In fact, the approach is dedicated to image because nodes represent keypoints in the image and node/edge features are extracted and learned to perform graph matching. The second step is the affinity computation of matrix K (see Model 6(RSGMIQP)). Since K is large, it is computed as in [Zhou and la Torre, 2016] thanks to a factorization of the matrix K. K is decomposed into smaller matrices: Node-edge incidence matrix, node-to-node similarity matrix and edge-to-edge similarity matrix. Then the relaxed graph matching (see Model 6(RSGMIQP)) is solved thank to a spectral method that computes the leading eigenvector of the matrix K. The solution y^* of the relaxed graph matching problem is further refined by adding L1 constraints (one-to-one mapping constraints) $\forall i, \sum_i y_{ik} = 1$ and $\forall k, \sum_k y_{ik} = 1$. This is performed by the Sinkhorn-Knopp algorithm [Sinkhorn and Knopp, 1967, Knight, 2007]. The fifth step goes back to the image by measuring the 2D-displacement d_i between two matched node i and k. Finally a loss is computed by computing a distance between d and the ground-truth displacement d^{gt} from the source point to the correct assignment. The key contribution is the construction of the different matrix layers, obtaining analytic derivatives all the way from the loss function down to the feature layers in the framework of matrix backpropagation.

The model is designed to establish correspondences between two images. The method scales up to affinity matrices K of size $10^6 \times 10^6$. The method is evaluated by the Percentage of Correct Keypoints metric.

4.1.1.3 Summary

Structural machine learning suffer from a main issue is how to confer learning capability. How to improve matching by learning? A hand-crafted matching cost function may perform poorly in practical problems. It means that the optimal solution, in term of graph matching objective function, does not lead to a low hamming distance with respect to the human ground-truth. A key idea is then to learn parameters of the matching cost function in order to better match two graphs G_1, G_2. A high level picture of the literature is provided in Figure 4.7. It is useful to capture at glance the paper distributions withing several families of methods. However, a more precise representation is needed to understand the new trends and issues.

[3] https://github.com/alexnowakvila/QAP_pt

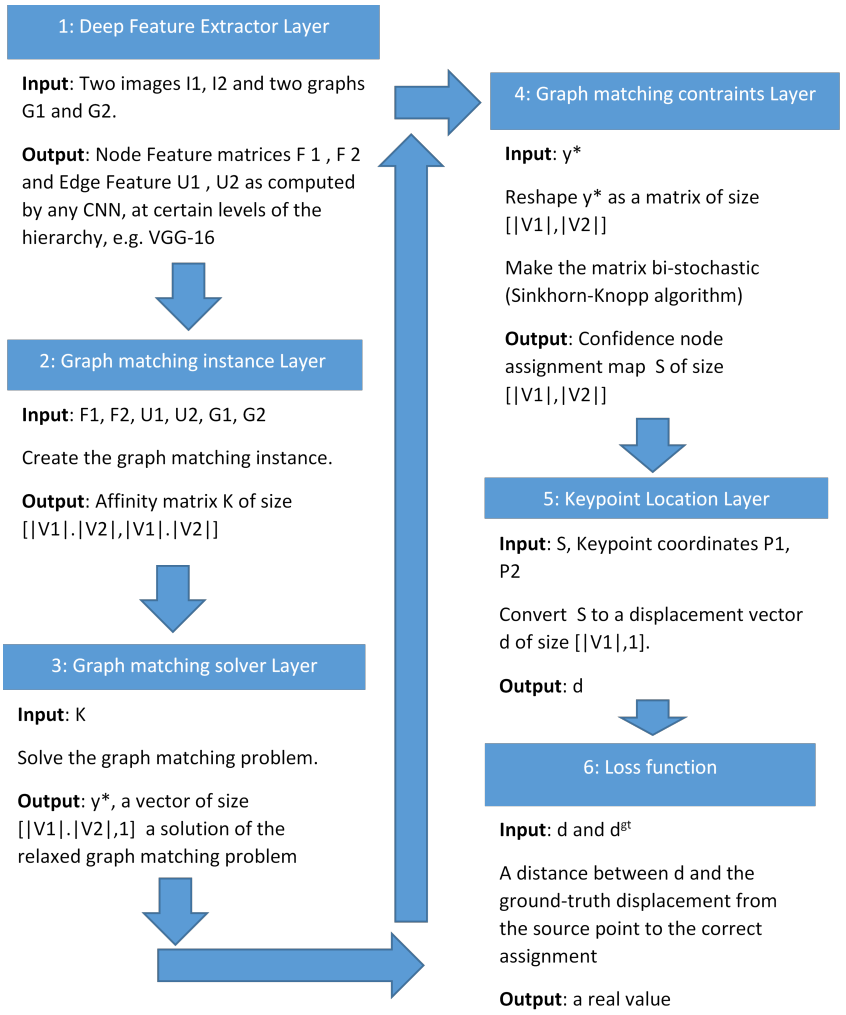


Figure 4.6: Computational pipeline of the fully trainable graph matching model.

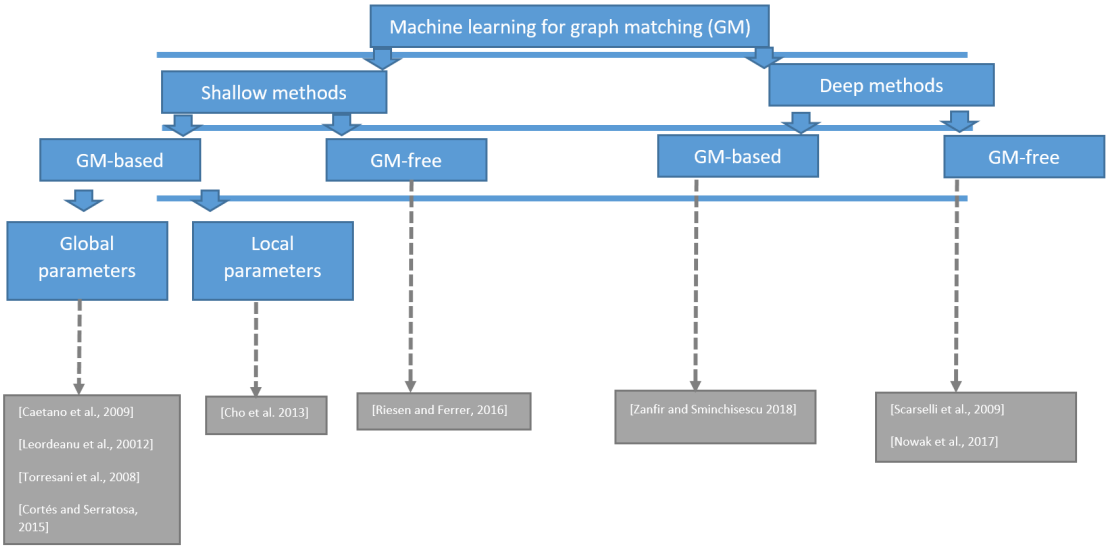


Figure 4.7: A literature review of machine learning techniques for graph matching.

The literature is organized with greater details according to the following criteria:

- The minimizer of the learning problem: This criterion corresponds to the learning algorithm used to minimize the loss function.
- Graph matching solver: The graph matching solver involved. Some approaches do not used any graph matching solver.
- The type of loss to be minimized.
- The parameterization level: Where are trainable parameters introduced? Are they global to the whole graph or local to nodes and edges?
- Distance or Feature Learning: Does the method learn node/edge distance? Or are node and edge features learned to better match?
- Are node/edge insertion and deletion considered explicitly?
- Supervised or unsupervised learning: Do the data set need to be labeled or ground-truthed and what kind of ground-truth is needed.
- Attribute types: Is a method benchmarked on richly attributed graphs (node/edge labelled with numeric vector, symbolic value, ...)?
- Scalability/Graph size: What is the graph size of the graphs involved in the experiments?

A literature summary is tabled in Table 4.1. Focusing on the learning problem, it is often solved by constraint quadratic programming or by gradient descent algorithms. Recently, the later is gaining importance thanks to deep neural networks and consequently the need to compute gradients efficiently is increasing.

Focusing on the GM module, the GM solvers are based on the QAP or the MAP-inference problems. All models are based on quadratic programming. However, the instances are often solved in the relaxed domain. At the opposite, some methods do not involve any matching solver.

Ref	Learning mini-mizer	GM solver	Loss	Parameter level	Feature Learning	Node/edge insertion	Supervised	Attribute type	Graph size
[Caetano et al., 2009]	Gradient descent	Mathematical solver	Hamming distance	Global	Features	No	Supervised	Numeric	200
[Leordeanu et al., 2012]	Gradient descent	spectral GM	Hamming distance	Global	Features	No	Unsupervised	Numeric	50
[Leordeanu et al., 2009]	Gradient descent	IPFP	Hamming distance	Global	Features	No	Supervised	Numeric	50
[Torresani et al., 2008]	Nonlinear Inverse Opti-mization	Dual Decom-position	?	Global	Distance	No	?	Numeric	100
[Cortés and Ser-ratosa, 2015]	Constraint Quadratic program-ming	BP	GED gap	Global	Distance	Yes	Supervised	Numeric	100
Cho et al. [2013]	Constraint Quadratic program-ming	Reweighted random walks for GM	Hamming distance	Local	Distance	Yes	Supervised	Numeric	100
[Riesen and Ferrer, 2016]	Constraint Quadratic program-ming	No	Hamming distance	Local	Distance	Yes	Supervised	Numeric	?
[Scarselli et al., 2009]	Gradient descent	No	Hamming distance	Local	Feature	No	Supervised	Nothing	?
[Nowak et al., 2017]	Gradient descent	No	Hamming distance	Local	Feature	No	Supervised	Nothing	50
Zanfir and Smin-chisescu [2018]	Gradient descent	Spectral GM	XY-displacement	Local	Feature	No	Supervised	Numeric	1000

Table 4.1: Synthesis of the literature on learning graph matching. The double horizontal lines separate between shallow and deep methods.

These methods rely on local statistics to capture the combinatorial nature of the graph matching problem.

The losses are mainly focused on the Hamming distance. A method ([Zanfir and Sminchisescu, 2018]) is specialized to work on images where graph nodes are 2D-points.

Chronologically speaking, first, methods focused on the global parameters by learning node and edge features globally over the entire graph. It means that global parameters are shared for all nodes and edges. Then, a set of local parameters that are node or edge dependent have been learned with success. First, the goal was to learn node-to-node distances and then the ambition was extended to node features and edge features learning. Recent methods go beyond graph matching where the two input graphs are given. The tendency is to learn the structure and the features to better match. None of the papers reported graph matching experiments with symbolic attributed graphs (meaning graphs with symbolic attributes). However, it does not mean that symbolic attributes are intractable for the methods. In the machine learning community, symbolic values, like words, letters for instance, are often represented by sparse one-hot vectors. Such an approach could be considered to deal with symbolic attributed graphs. Graph sizes range from 50 nodes to 1000 nodes. Graph matching based on deep learning is not completely mature in terms of performance evaluation. Data sets are often made of synthetic graphs. For the shallow methods, the experimental parts is often much stronger with a clearer comparison between learning-free and learning-based methods. A comparison of graph matching methods with and without learning are provided in Table 4.2 and Table 4.3. Table 4.2 presents results where an unsupervised, learning procedure improves accuracy of the two graph matching methods Graduated Assignment (GA [Gold and Rangarajan, 1996]) and Spectral Matching (SM [Leordeanu and Hebert, 2005]). In the experiment the spectral graph matching takes more advantage of the learning scheme. At the opposite, Table 4.3 reports performance evaluation of a supervised learning method against a learning-free method. Accuracy improvements are observed (around 10%-15%) for the learning-based methods against the learning-free method. When the gain in accuracy thanks to learning

Dataset	Matching methods SM	GA
Cars: no learning	26.3%	31.9%
Cars: with learning	62.2%	47.5%
Motorbikes: no learning	29.5%	34.2%
Motorbikes: with learning	52.7%	45.9%

Table 4.2: Taken from [Leordeanu et al., 2012]: Comparison of matching rates for 2 graph matching algorithms before and after unsupervised learning on Cars and Motorbikes from Pascal07 database, with all outliers from the right image allowed and no outliers in the left image.

Learning methods	Matching methods SM Accuracy (%)	IPFP Accuracy (%)
w/o learning	60.4	61.4
DW-SSVM [Cho et al., 2013]	66.2	69.0

Table 4.3: Taken from [Cho et al., 2013]: Performance on synthetic point sets. A learning approach and a learning-free method (shown in each row) are evaluated with the state of the art graph matching algorithms (in columns)

methods seem well evaluated, the impact on the convergence speed of the learning algorithm is not discussed. Running time are rarely reported in the experiments so it is hard to draw conclusion about it. However, in the deep learning architecture of [Zanfir and Sminchisescu, 2018], for graphs with 1000 nodes, a complete forward and backward pass runs in roughly 2 seconds on a 3.2 Ghz Intel Xeon machine, with Titan X Pascal GPU.

Finally, the amount of data to do the training is not really discussed. In [Cho et al., 2013], for images of identical objects, such as these House/Hotel sequences, only a few number of images are sufficient for the method to learn the model graph and its features.

4.1.2 Open problems

In some applications for instance Computer Aided Drawing, the graph G_2 can be a perfect model of a symbol and G_1 a noisy graph. However, in realistic situations, the source of information cannot be accessed directly (i.e. there is no ideal model graph available). So, the model graph can only be sampled from the source (the data set). Although it is useful to learn a matching function for two graphs of a certain class, a more appealing goal would be to learn a graph model to match, which provides an optimal matching to all instances of the class. Such a learned graph would better model the inherent structure in the target class, thus resulting in better performance for matching. So the new question is merging graph learning and graph matching: How to obtain a graph model for matching ? This question that has been discussed in [Cho et al., 2013, Zanfir and Sminchisescu, 2018]. Another way of thinking is to explicitly learn insertion/deletions costs to cope with noisy model graphs. If a model graph G_2 holds unnecessary vertices then their insertion costs should be small. This was not investigated yet. Other interesting questions are still opened about the accuracy of predicted matchings. Is it possible to parametrize an heuristic method to reach solutions close the optimal solutions? Can we learn to approximate an exact method? Can we speed up an heuristic method thanks to trained cost function? Does a trained cost function makes instances easier to solve? These questions lead to the following deadlocks:

1. Deadlock 8 : How to deal at a fine level with insertion and deletion costs? Can we model

noisy referent graph thanks to these costs.

2. Deadlock 9 : Can an heuristic output solutions closer to optimality thanks to machine learning?

These two deadlocks will be discussed in the next section. In addition, from the literature review, wider problems appear:

1. How to deal with symbolic attributes? Are one-hot vector a possible solution?
2. What is the amount of data required to do the learning?
3. Benchmarking, common data sets and evaluation protocol.
4. A better understanding of the training impact, generalization power and size of the training set.

These questions and remarks will be developed in the perspective section (Section 5) of this manuscript.

4.1.3 Contributions

4.1.3.1 Deadlock 8 : How to deal at a fine level with insertion and deletion costs?

4.1.3.1.1 Motivation In the context of the ECGM problem, an interesting question arises: Can insertion/deletions cost function be learned to cope with noisy sample graphs? How to parametrize the graph edit distance to learn node/edge dependent insertion/deletion costs? Starting from the disrcrimitative weight formulation proposed in [Cho et al., 2013], we proposed to extend it to take into account also insertion/deletions costs. Finally, in our proposal, the learning problem is not solved by constraint quadratic programming method like in [Cho et al., 2013] but by gradient descent.

4.1.3.1.2 Details Let $G_1 = (V_1,E_1,\mu_1,\zeta_1)$ and $G_2 = (V_2,E_2,\mu_2,\zeta_2)$ be two graphs, with $|V_1| = n_1$ and $|V_2| = n_2$. To apply removal or insertion operations on nodes, node sets are augmented by dummy elements. The removal of each node $i \in V_1$ is modeled as a mapping $i \to \epsilon_{2,i}$ where $\epsilon_{2,i}$ is the dummy element that is associated with i. As a consequence, the set V_2 is increased by n_1 dummy elements ϵ_2 to form a new set $\mathcal{V}_2 = V_2 \cup \epsilon_2$. The node set V_1 is increased similarly by n_2 dummy elements ϵ_1 to form $\mathcal{V}_1 = V_1 \cup \epsilon_1$. Note that $\mathcal{V}_1$ and $\mathcal{V}_2$ have the same cardinality : $N = n_1 + n_2$. Each element of the two graphs can be edited only once.

Now, we explain how we parameterize Equation 4.2. Let $\pi(k) = i$ denote an assignment of node $k \in G_2$ to node $i \in G_1$, i.e. $y_{ik} = 1$. A joint feature map $\Phi(G_1; G_2; y)$ is defined by aligning the relevant dissimilarity values of Equation 3.2 into a vector form as: $\Phi(G_1, G_2, y) = [\cdots, d_V(\pi(k), k), \cdots, d_E((\pi(k), \pi(l)), (k, l)), \cdots]$. By introducing a real-valued vector β to weight all elements of this feature map, we obtain a discriminative objective function:

$$d(G_1, G_2, y, \beta) = \beta^T \Phi(G_1, G_2, y) \tag{4.12a}$$

$$= \cdots + \beta_k \cdot d_V(\pi(k), k) + \\ + \beta_{kl} \cdot d_E((\pi(k), \pi(l)), (k, l)) + \cdots \tag{4.12b}$$

where β is a weight vector that encodes the importance of node and edge dissimilarity values. In the case of uniform weights, i.e. $\beta = \mathbf{1}$, all elements of vector β are 1, and Equation 4.12a is reduced to the conventional graph matching score function of Equation 3.2: $d(G_1, G_2, y) = d(G_1, G_2, y, \mathbf{1})$. An

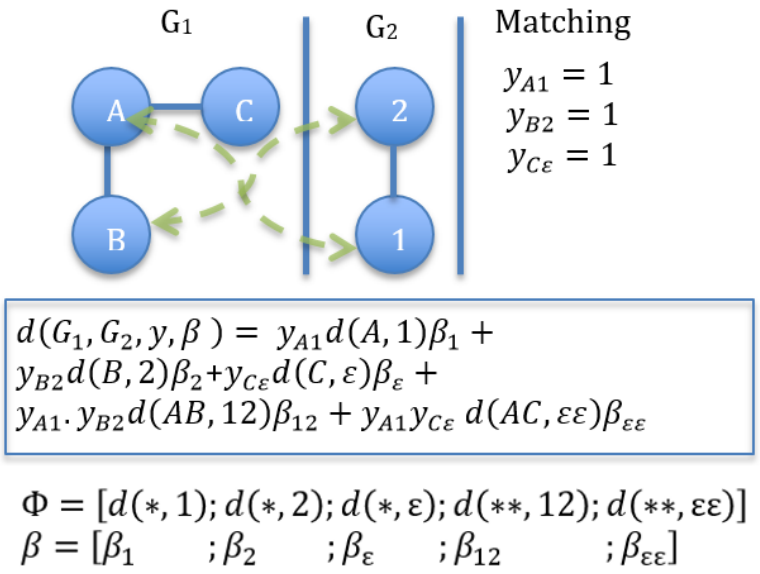


Figure 4.8: Illustration of the parametrized score function computation.

example of the parametrized objective function is given in Figure 4.8. The discriminative weight formulation is general in the sense that it can assign different parameters to individual nodes and edges. However, it does not learn a graph model that underlies the feature map, and requires a reference graph G_2 at query time, whose attributes cannot be modified in the learning phase. The new discrete optimization problem can be rewritten from Model GMIQP as follows:

Model 7. *Parametrized graph edit distance (PGED)*

$$y^* = \underset{y}{\operatorname{argmin}} \quad d(G_1, G_2, y, \beta) \tag{4.13a}$$

$$\textit{subject to} \quad y \in \{0,1\}^{N.N} \tag{4.13b}$$

$$\sum_{i=1}^{N} y_{i,k} = 1 \quad \forall k \in [1, \cdots, N] \tag{4.13c}$$

$$\sum_{k=1}^{N} y_{i,k} = 1 \quad \forall i \in [1, \cdots, N] \tag{4.13d}$$

Fixed parametrized graph matching for machine learning To perform conventional machine learning techniques such as Support Vector Machines (SVM) or Deep Neural Networks (DNN) on a data set, a fixed feature vector size is often mandatory. In the context of machine learning, G_2 is the model graph and is considered to be fixed during the learning phase. Consequently, G_2 is renamed G_m in the manuscript. To obtain a fixed size, in Equation 4.12b, the sizes of the vectors β and $\Phi(G, G_m, y)$ must only depend on the size of G_m and not on the query graph G.

The graph elements of G and the components of graph $G_m = (V_m, E_m)$ are aligned into a vectorial form by the function Φ. $\Phi(G, G_m, y)$ is a vector $\in \mathbb{R}^{|V_m|+|E_m|+2}$. Two extra components

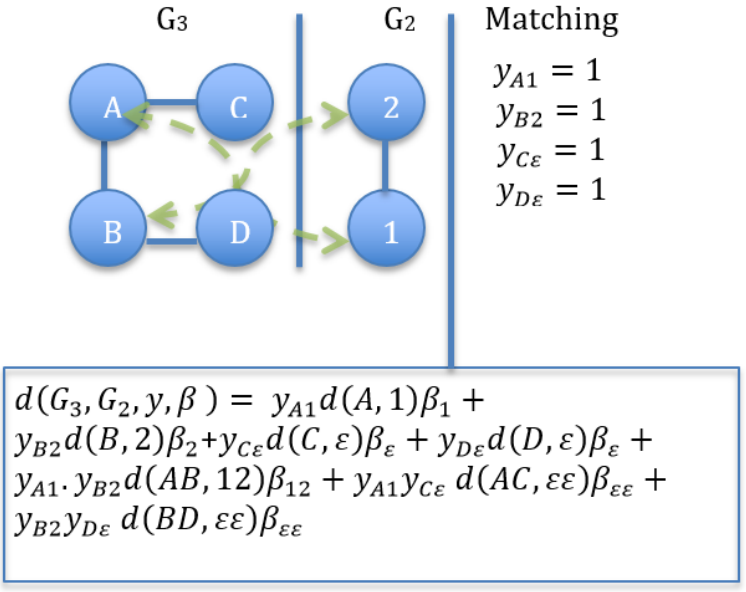


Figure 4.9: Parametrized matching function where G^1 has 7 graph components (4 nodes and 3 edges).

are added to accumulate node and edge deletion costs. For instance, it can occur that a node or an edge of G is mapped to a dummy element ϵ (node deletion or edge deletion). In such a case, the node deletion costs are mapped in the penultimate component of the vector $\Phi(\cdot)$. The edge deletion costs are mapped in the last component of the vector $\Phi(\cdot)$.

β is a vector $\in \mathbb{R}^{|N_m|+|E_m|+2}$. Individual weights are associated with each component of G_m. Two extra components are added to parametrize the node and edge deletion costs, respectively. A shared weight is associated with all node deletion costs. The same strategy is applied for edge deletions.

The value of the objective function remains the same. Substitutions and insertions have individual weights. Deletions share a mutual parameter. In Figure 4.8, G_1 holds 5 graph components and in Figure 4.9, G_1 holds 7 graph components. However, in both figures $\Phi(\cdot)$ and β are vectors of size 5 that only depend on model graph G_2.

Now that we have a parametrized version of the error-correcting graph matching problem. We can use it in a learning scheme that it explained in the next section (Section 4.1.3.2).

4.1.3.2 Deadlock 9: Can an heuristic output solutions closer to optimality thanks to machine learning?

4.1.3.2.1 Motivation Heuristics are known to be fast (polynomial time) but their solutions can be far from the optimal solutions. The key idea is to take advantage of fast heuristics but to use machine learning to increase their effectiveness. Effectiveness is to be understood in terms of matching accuracy.

4.1.3.2.2 Details Starting, from the parametrized graph edit distance presented in Model 7(PGED), we use it to predict the optimal matchings between input graphs G and a fixed model graph G_m. Optimal matchings are computed thanks to an exact algorithm (F2 in our case).

Optimal matchings are the target values of the learning algorithm so they are denoted by y^{gt}. Therefore, $TrS = \{(G_k, G_m), y_i^{gt}\}_{i=1}^{M}$ and we want to predict y. From the viewpoint of machine learning this problem falls into a structured regression problem also called structured prediction. The term "structured" refers to the fact that the output of the predictor is not a simple scalar value but a structured output (see Appendix B). Thanks to the joint-feature map Φ defined in the previous section, we can write structured prediction problem :

$$\min_{\beta, y} \sum_{(G,G_m), y^{gt} \in TrS} l(G, y^{gt}, G_m, y, \beta) \tag{4.14}$$

$$l = \frac{1}{2}(\Phi(G, G_m, y^{gt}) - \Phi(G, G_m, y))^2 \tag{4.15}$$

The learning problem is a minimization problem where appears both variables β and y. Both are involved into the minimization of an empirical risk guided by the loss function l. However, β is wanted to be invariant across the whole data set while y is computed for every input graph. The goal is to produce a feature map $\Phi(G, G_m, y)$ that is close to the ground-truth feature map $\Phi(G, G_m, y^{gt})$.

4.1.3.2.2.1 Learning algorithm One of the easiest ways to understand algorithms for general structured prediction is the structured perceptron of Collins [Collins, 2002]. This algorithm combines the perceptron algorithm for learning linear regressor with an inference algorithm (classically the Viterbi algorithm when used on sequence data) and can be described abstractly as follows. First define a "joint feature function" $\Phi(x, y)$ that maps a training sample x and a feasible solution y to a vector of length d (x and y may have any structure; d is problem-dependent, but must be fixed for each model). Let GEN be a function that generates a set of feasible solutions and α is the learning rate.

- Let β be a weight vector of length d.
- For a pre-determined number of iterations:
  - For each sample x in the training set with true output y^{gt}:
    - Find a feasible solution $y^* = \underset{y \in GEN(x)}{arg\,min}\ (\beta^T \Phi(x, y))$
    - Update β, from y^* to y^{gt}: $\beta = \beta + \alpha(-\Phi(x, y^*) + \Phi(x, y^{gt}))$

Now, let us design our perceptron-based learning algorithm for graph matching. Algorithm 2 is a deterministic algorithm. $\#iter$ is the maximum number of iterations or also called *epochs* in the literature. The parametrized graph matching problem is solved in Line 9. Line 10 applies the learning rule defined in Equation 4.16. To show the time-dependence of β, we use $\beta(i)$ as the weight at time i.

4.1.3.2.2.2 Learning rule. The learning rule aims at modifying β. The weights should be updated in cases of wrong predictions. The correction must take into account the amount and the sign of the committed error.

$$\begin{aligned} \beta(i+1) &= \beta(i) - \alpha \frac{\partial l(G, G_m, y^{gt})}{\partial \beta(i)} \\ \beta(i+1) &= \beta(i) - \alpha \left((\Phi(y^*) - \Phi(y^{gt})).\Phi(y^*) \right) \end{aligned} \tag{4.16}$$

This rule is obtained by deriving the the computation graph shown in Figure 4.10 with respect to parameters β.

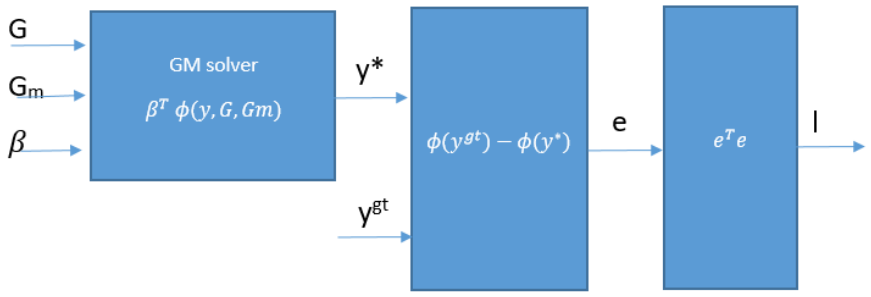


Figure 4.10: Computation graph of our learning problem from the inputs to the loss.

The calculus of the derivation goes as follows :

$$\frac{\partial l}{\partial e} = e \quad ; \frac{\partial e}{\partial y^*} = -1 \quad ; \frac{\partial y^*}{\partial \beta} \approx \Phi(y^*) \tag{4.17a}$$

$$\frac{\partial l}{\partial \beta} \approx \frac{\partial l}{\partial e} . \frac{\partial e}{\partial y^*} . \frac{\partial y^*}{\partial \beta} \tag{4.17b}$$

$$\frac{\partial l}{\partial \beta} \approx -e.\Phi(y^*) \tag{4.17c}$$

$$\frac{\partial l}{\partial \beta} \approx -(\Phi(y^{gt}) - \Phi(y^*)).\Phi(y^*) \tag{4.17d}$$

$$\frac{\partial l}{\partial \beta} \approx (\Phi(y^*) - \Phi(y^{gt})).\Phi(y^*) \tag{4.17e}$$

$$\tag{4.17f}$$

Parameters are updated in the opposite direction of the gradient. The goal is to get closer and closer to the best parameter values β_{min} as shown in Figure 4.11. We decided to not propagate gradients through the graph matching method (inside each iteration of the solver) but we do it only once at convergence ($\frac{\partial y^*}{\partial \beta}$). We are aware of that shortcut. The goal was to be faster maybe at the cost of a noisier or unstable gradient.

Algorithm 2 Training the graph-based perceptron for matching.

INPUT: $TrS = \{(G_k, y_k^{gt})\}_{k=1}^{M}$ and G_m
INPUT: $\#iter$ is the maximum number of iterations
INPUT: α learning rate
OUTPUT: Learned β. A weight vector
Init: $\beta \leftarrow \mathbf{1}$ and $iter \leftarrow 0$
while iter< $\#iter$ **do**
  i $\leftarrow 0$
  for $(G, y^{gt}) \in TrS$ **do**
    $y^* \leftarrow \underset{y \in \Gamma(G, G_m)}{\arg\min} \quad \beta(i)^T \cdot \Phi(G, G_m, y)$ `// Solve Model 7(PGED)`
    $\beta(i+1) = \beta(i) - \alpha\,((\Phi(y^*) - \Phi(y^{gt})).\Phi(y^*))$
    i $\leftarrow$ i $+1$
  end for
  iter $\leftarrow$ iter $+1$
end while

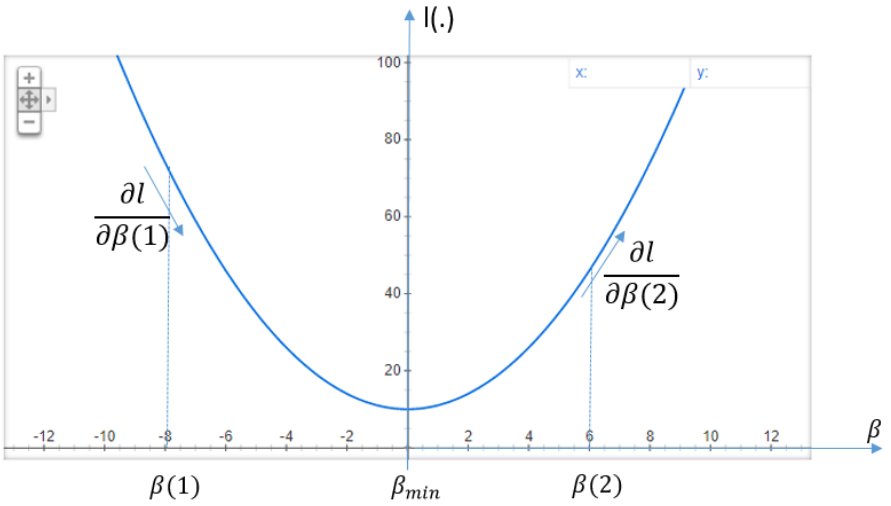


Figure 4.11: Principle of the gradient descent.

Methods	Accuracy
Optimal	1
BP learned	0.58
BP w/t learning	0.48

Table 4.4: Test: Mean matching accuracy on unseen graphs.

4.1.3.2.2.3 Experiments Algorithm 2 requires a GED heuristic. The method called BP [Riesen and Bunke, 2009] has been chosen because it is fast but other could be applied. Algorithm 2 was run on CMU House data set. The model graph is the k^{th} graph of the data set and k varies from 1 to 100. Nodes have no attributes and edges convey information about the distance between two points. Distance between two edges d_E is the L1 norm between two scalar values. Such graphs represent a challenging task for the matching algorithms. Three methods are considered. 1) The matching given by an exact method (optimal matching), 2) and 3) The matching provided by BP with and without learning. For the learning free method, the matching costs were taken from the GED contest [Abu-Aisheh et al., 2017a]. Half of the data set was used for training and the other half for testing. Results of the Algorithm 2 are reported in Table 4.4 and Figure 4.13. In Figure 4.13, the average matching accuracy is depicted in function of the iterations of the algorithm. The learning scheme can improve the matching accuracy by 25% in average on the training set. Results obtained during the test phase are reported in Table 4.4, the gain is about 20%. The gain is smaller due to the difficulty to generalize on unseen graphs. However, results are still far from the exact method in terms of accuracy but the heuristic remains much faster. On CMU instances, the heuristic had a running time about 80 ms when the exact method took 1500 seconds. In Figure 4.12 graph matching results are presented for a given pair of graphs at iteration 0 and iteration 200.

Finally, a main drawback of such an approach is that parameters are dependent on the model graph G_m. Parameters must be found and stored for each model graphs. Another issue is that this method does not learn the features but only the local dissimilarities. The β weight vector could be extended to be a matrix in order to weight each feature component of a given node or edge.

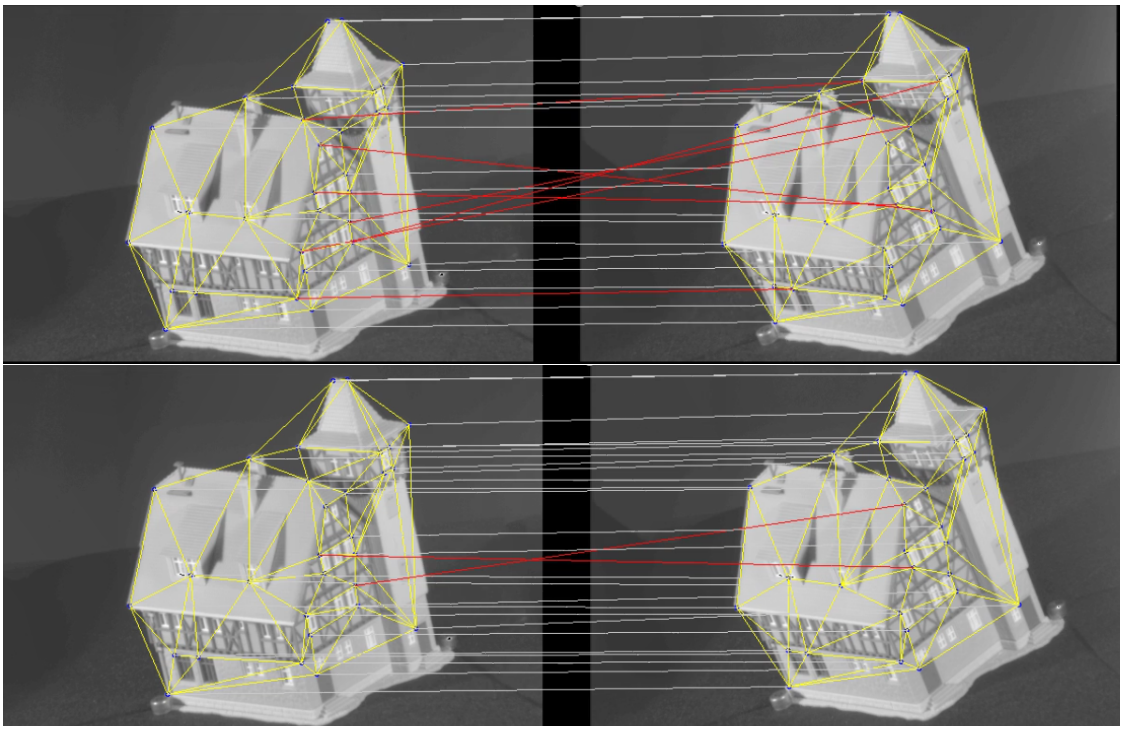


Figure 4.12: Matching evolution from iteration 0 (top) to 200 (bottom).

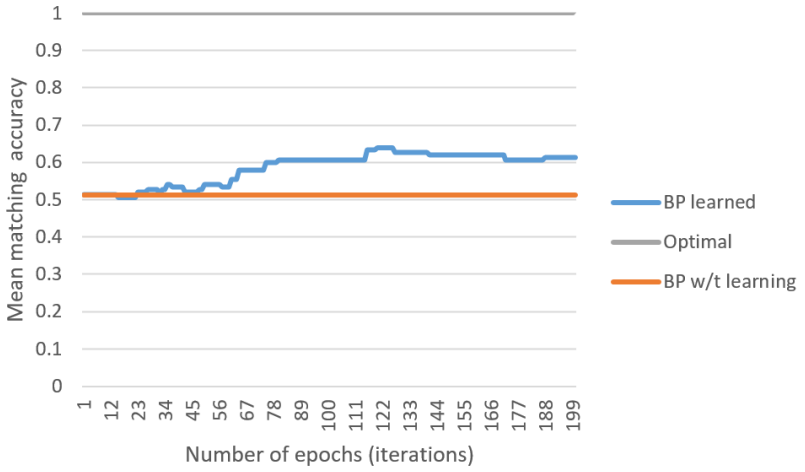


Figure 4.13: Training: Accuracy in function of the number of epochs.

4.1.4 Summary

During the PhD of Maxime Martineau, we have worked on parametrizing the GED problem to cope with noisy reference graphs [Raveaux et al., 2017]. Then, a learning scheme was designed to include the parametrized GED formulation. The goal to be achieved by the learning scheme is to approximate an exact GED solver.

Now let us outline some drawbacks. Learning from examples (supervised learning) might be undesirable for $\mathcal{NP}$-hard problems because (1) the performance of the model is tied to the quality of the supervised labels, (2) getting high-quality labeled data is expensive and may be infeasible for some instances, (3) one cares more about finding a competitive solution more than replicating the results of another algorithm. This last statement opens the door to a whole area of research. It will be discussed in the perspectives.

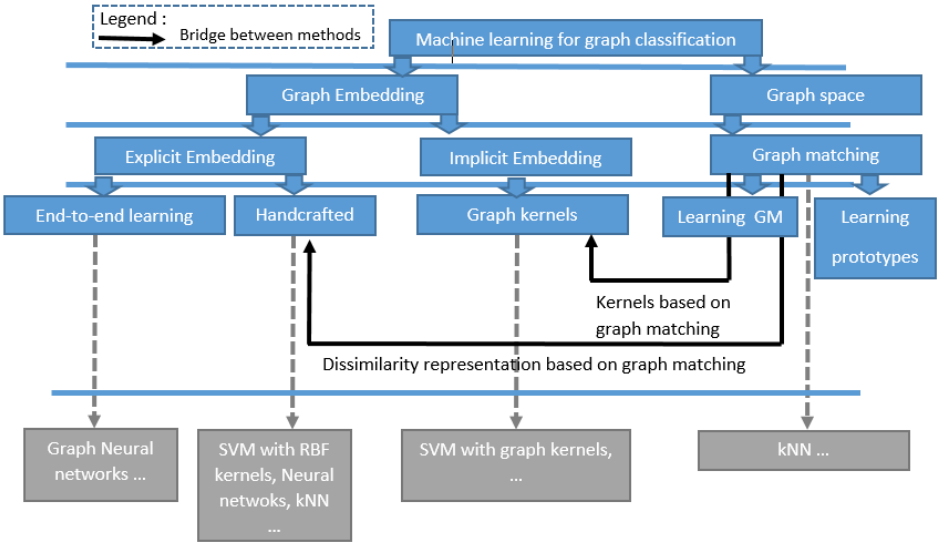


Figure 4.14: Machine learning techniques for graph classification and a focus on bridging the gap between graph matching and embedding techniques.

4.2 Graph classification

4.2.1 State of the art from a machine learning viewpoint

As mentioned in Section 2.3, graph classifiers can be categorized into two categories whether the classifier operates in a vector space or in a graph space. Vector space methods are fast but efficiency comes at a price: feature vector transformation leads to loss of topological information. The graph space paradigm is characterized by the fact that graphs are compared in a graph space. Whereas, this definition was straightforward in a learning free context (Section 3.2). In a machine learning context, the boundaries are less clear. So, we come up with a definition of machine learning in graph space as follows:

Definition 12. *Machine learning in graph space*
Machine learning techniques that rely on graph matching to extract features.

This manuscript is focused on the methods that respect this definition.

Typically, kNN, kernel machines and explicit embeddings can be compliant with Definition 12 if they rely on graph matching. Figure 4.14 draws a picture at a coarse level of machine learning techniques for graph classification. In addition, Figure 4.14 illustrates the links between techniques operating in graph space and vector space.

Let us comment this important statement thanks to Table 4.5 where the families of classification methods are categorized according to their compliance to Definition 12.

Both graph kernels and graph embeddings provide a powerful vectorial description of the underlying graphs. While graph kernels produce an implicit embedding of graphs into a Hilbert space, graph embeddings result in an explicit feature vector in a real vector space. Yet, both approaches crucially depend on similarity or dissimilarity computation on graphs. In Appendix D, a review of kernels based on graph matching is provided. In a classification context, the training set is defined as $TrS = \{(G_i, t_i)\}_{i=1}^{M}$, where $G_i \in \mathcal{D}$ is a graph and $t_i \in \mathcal{T}$ is the class of the graph. One of the significant limitation of (dis)similarity based algorithms is that the kernel or distance functions

Machine learning in graph space	Family of methods
No	Explicit graph embedding [Luqman et al., 2013]: each graph is mapped to a feature vector ($\phi : \mathcal{G} \rightarrow \mathbb{R}^n$). Thereafter it is not trivial to return in a graph space. The function ϕ is explicitly defined. By analogy with the image processing field, such methods can be seen as handcrafted methods to extract features from graphs. Classifiers are trained on vectors.
No	Graph kernels [Gärtner, 2003]: They are similarity measures corresponding to a scalar product in a vector space that is not necessary known explicitly ($k :< \mathcal{G}, \mathcal{G} >\rightarrow \mathbb{R}$)). Each pair of graphs is compared thanks to substructures that are computable in polynomial time. Substructures are used to compute the scalar product of the graph pair. The kernel must encode a measure of relevant similarity between the graphs while limiting the complexity of calculation and respecting the different properties defining a kernel. Classifiers are trained in a kernel space.
No	Graph neural networks [Kipf and Welling, 2016]: They can be categorized as explicit graph embedding techniques but the function ϕ is not handcrafted but learned.
Yes	The graph kernel is based on GED (i.e. $k(G_1, G_2) = exp(-GED(G_1, G_2))$) [Neuhaus and Bunke., 2007]. kNN or SVM classifiers can be employed for instance. In such a case, graph comparisons are performed in a kernel space but the kernels rely on graph matching.
Yes	kNN based on GED [Riesen, 2015].
Yes	The explicit graph embedding defined by $\phi(G) = [GED(G,G_1), \cdots, GED(G,G_M)]$ [Riesen and Bunke, 2010b]. In this case, the graph comparisons are performed in a vector space. However, each component of the vector is the result of a graph matching algorithm.

Table 4.5: Categorization of high level methods for comparing graphs.

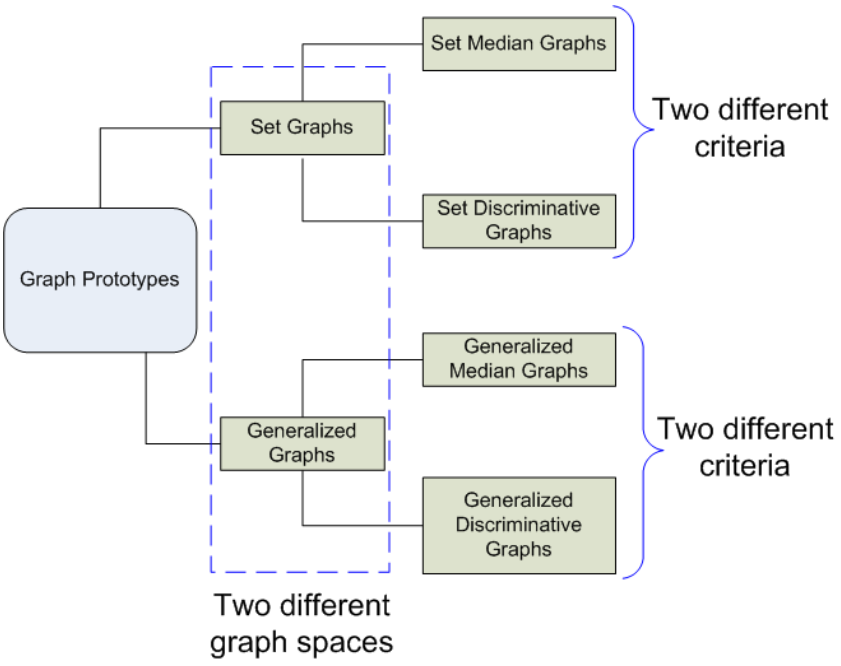


Figure 4.15: Taxonomy of graph prototypes.

must be evaluated for all possible pairs (G, G_i) where G is a graph from the test set and G_i is a graph from the training set. All these evaluations can lead to excessive computation times when making predictions for new graphs. So, from this statement arises a new problem how to reduce the training set to make the decision faster. This problem is referred in this dissertation as **learning graph prototypes**.

4.2.1.1 Learning graph prototypes

Graph prototypes can be categorized according to the space they belong to. Prototypes that necessarily belongs to the training set are called "*set graphs*". At the opposite, prototypes that do not necessarily belong to the training set are called "*generalized graphs*". Generalized graphs can be artificially generated and do not necessarily represent real objects. Each prototype family can then be refined according to the computation criteria. We distinguish between generative and discriminative prototypes. Generative prototypes model the data distribution ($Pr(\mathcal{D})$ or $Pr(\mathcal{D}|t)$) while discriminative prototypes maximize a classification rate (equivalent to the conditional probability $p(t|\mathcal{D})$) if t is the ground-truth class label. Generative prototypes are obtained in a unsupervised manner. However, such prototypes do not take into account the inter-class distribution of learning samples. Discriminative prototypes require a supervised scheme to be computed.

The Figure 4.15 pictures out the graph prototype taxonomy.

The objectives of graph prototypes are (i) to overcome the well-known disadvantages of a (dis)similarity-based classifier, i.e. the large storage requirements, the large computational effort and the sensitivity to noisy examples and (ii) to keep classification performance as high as possible.

4.2.1.1.1 Set prototypes SVM [Vapnik, 1998, Neuhaus and Bunke., 2007] based methods have sparse solutions, so that predictions for new inputs depend only on the similarity/kernel function evaluated at a subset of the training data. In Appendix D, an introduction to SVM is presented to illustrate this principle.

In order to classify new graph using the SVM trained model, the sign of the classifier $f(G)$ is expressed in terms of the Lagrange multipliers $\{a_n\}$ and the kernel function. $\{a_n \in \mathbb{R}^+\}$ are parameters and they are found by solving the dual formulation of SVM (Equation D.8). According to a training set $TrS = \{(G_i, t_i)\}_{i=1}^{M}$ where t is the class label, the classification is then expressed by :

$$\hat{t} = \sum_{i=1}^{M} a_i t_i k(G, G_i)$$

$\hat{t}$ is the predicted class. Any graph for which $a_n = 0$ will not appear in the sum in and hence plays no role in making predictions for new graphs. The remaining graphs are called *support vectors* and because they satisfy $t.\hat{t} = 1$, they correspond to graphs that lie on the hyperplane of separation between classes. This property is central to the practical applicability of support vector machines. Once the model is trained, a significant proportion of the training data can be discarded and only the support vectors are retained.

$$\hat{t} = \sum_{i \in S} a_i t_i k(G, G_i)$$

Where S denotes the set of indices of the support vectors. According to the taxonomy explained in Figure 4.15, SVM-based prototypes are discriminative set prototypes. Similarly to SVM, in [Borzeshi et al., 2013], another framework for selecting a set of prototypes is proposed. It selects graphs from a labelled graph set taking their discriminative power into account.

Another kind of set prototypes are set median graphs [Ferrer et al., 2009, 2011, Jiang et al., 2001]. In a classification context, median graphs are computed independently in each class through a minimization process of the sum of distances to all graphs within a given class. The set median graph for the class i is defined as follows:

Definition 13. *Set median graph*
Let $S = \{Trs | t = i\}$ be the set containing all the graphs of the class i.

$$smg = arg \min_{(G_2,t) \in S} \sum_{(G_1,t) \in S} d(G_1, G_2)$$

The concepts presented above involve the generation of a single prototype for each class. In some particular applications, it may be interesting to generate m prototypes for each class in order to obtain a better description of data. In what follows, we give the definition of such prototypes called $msmg$ [Raveaux et al., 2011]:

Definition 14. *Multiple set median graphs*
Let $S = \{Trs | t = i\}$ be the set containing all the graphs of the class i. Let $Proto = \{G_k\}_{k=1}^{m} \subset S$ be a subset of S with m graphs. Let $d_{min}(G, Proto)$ be the smallest distance between a graph G and $Proto$ defined as follows:

$$d_{min}(G, Proto) = \min_{G_j \in Proto} d(G, G_j)$$

Then, the set of multiple median graphs is defined by:

$$msmg = arg \min_{Proto \subset S} \sum_{G_1 \in S} d_{min}(G_1, Proto)$$

The problem of finding the $msmg$ is known in the literature as the $p-median$ problem. The $p-median$ problem is a combinatorial problem known to be $\mathcal{NP}$-hard [Mladenovic et al., 2007]. A variation of the $msmg$ are the spanning set graphs ($mspg$) [Riesen and Bunke, 2010a]. Contrary to multiple median graphs, the spanning prototypes can be computed in polynomial time. This set of graphs is built by iterations. At the first iteration, the first prototype selected is the set median graph and then each additional prototype is selected by the spanning prototype selector. The spanning prototype selector is the graph the furthest away from the already selected prototype graphs. Such a graph set can be defined as follows:

Definition 15. *Spanning set graphs*
Let $S = \{Trs|t = i\}$ be the set containing all the graphs of the class i. Let $mspg = \{G_k\}_{k=1}^{m} \subset S$ be a subset of S with m graphs. Let $d_{min}(G, mspg)$ be the smallest distance between a graph G and $mspg$ defined as follows:

$$d_{min}(G, mspg) = \min_{G_j \in mspg} d(G, G_j)$$

Then, the set of spanning graphs can be iteratively constructed as follows:

$$mspg_j = \begin{cases} smg & if\ j = 1 \\ mspg_{j-1} \cup \{G_j\} & if\ 1 < j \leq m \quad where\ G_j = arg \max_{G \in \{S \setminus mspg_{j-1}\}} d_{min}(G, mspg_{j-1}) \end{cases}$$

$mspg_j$ denotes the set at iteration j.

Finally set prototypes can also be obtained by any distance based clustering algorithm. For instance, the PAM algorithm [Kaufman and Rousseeuw, 1987] is a clustering algorithm operating on a distance matrix to compute medoids. In contrast to the k-means algorithm, PAM can be used with arbitrary distances.

4.2.1.1.2 Generalized prototypes The generalized median graph of a set of graph S is a graph that minimizes the sum of the distances to all graphs in S. The generalized median graph differs from the set median graph because it does not necessary lie in S [Musmanno and Ribeiro, 2016, Ferrer et al., 2010, Chaieb et al., 2017]. It is defined by :

Definition 16. *Generalized median graph*
Let U be the infinite set of graphs that can be built using the labels from L_V and L_E. Let $S = \{G_1, \cdots, G_M\} \subset U$ be a subset of U. The generalized median graph (gmg) of the subset S is defined by:

$$gmg = arg \min_{G_2 \in U} \sum_{G_1 \in S} d(G_1, G_2)$$

Using median graphs, essential information of each class is captured. However, such prototypes do not take into account the inter-class distribution of learning samples. In order to overcome this problem, discriminative graphs [Raveaux et al., 2011] (dg) has been proposed as prototypes for graph classification. The main difference between median graphs and discriminative graphs lies in the criterion which is used to generate the prototypes. In the case of dg, rather than optimizing a sum of intra-class distances, prototypes are generated in order to optimize the classification error rate obtained on a test dataset.

Definition 17. *Generalized discriminative graphs*
Let N be the number of classes of the classification problem. Let TrS be a training set and let $\Delta(TrS, \{G_i\}_{i=1}^{N})$ be the error rate obtained by a 1NN classifier on TrS using the graph prototypes $\{G_i\}_{i=1}^{N} \subset U$ as learning samples. Then the set GDG composed of the gdg of each class is given by:

$$\begin{aligned} GDG = & \quad \{sdg_1, \cdots, sdg_N\} \\ GDG = & \quad arg\ \min_{\{G_i\}_{i=1]^N} \in U} \Delta(TrS, \{G_i\}_{i=1]^N}) \end{aligned}$$

The concepts presented above involve the generation of a single prototype but it can be extended to m prototypes.

Note that in [Cho et al., 2013], generalized prototypes are learned. However, the criterion to generate them is to maximize a matching rate and not a classification rate.

As it is explicitly mentioned in Definitions 13, 14, 15, 16, 17, a (dis)similarity measure is required to compute prototypes. However, as discussed in Section 4.1, the (dis)similarity measure can also be learned to fit a specific objective. So, the next section is devoted to this topic called "Learning graph (dis)similarity measure".

4.2.1.2 Learning graph (dis)similarity measure

Another room, where learning algorithms can be introduced, is the learning of the (dis)similarity measure. This topic is often called metric learning. Metric learning can be achieved by many means such that :

- learning kernel parameters [Gärtner, 2003]
- learning a graph embedding [Riba et al., 2018]
- learning a distance in the vector space where the graphs are projected
- metric learning for structured data [Collins, 2002]

However, we may focus on metric learning for structured data to be compliant with the Definition 12(LGM). Note that when kernels rely on graph matching, learning kernel parameters can also be compliant with Definition 12(LGM). However, kernels based on graph matching need cost functions. So the question of learning graph matching cost function is still crucial. In [Riesen and Bunke, 2009], a grid search on a validation set is used to determine the values of the parameters $K_V \in \mathbb{R}$, which corresponds to the cost of a node deletion or insertion, and $K_E \in \mathbb{R}$, which corresponds to the cost of an edge deletion or insertion. The main drawback of the grid search is the empirical definitions of intervals and bounds of the grid, which require expertise on the problem. The speed can also drastically decrease as the grid becomes larger in function of the size parameter space. The method aimed at learning common weights for all the edges and nodes and only distinguishes weights by type of operations: deletion, insertion or substitution. Neuhaus et al. [Neuhaus and Bunke, 2005] address the issue of learning dissimilarity functions for numerically labeled graphs from a corpus of sample graphs. A system of self-organizing maps (SOMs) that represent the dissimilarity spaces of node and edge labels was proposed. The learning process adapts the edit costs in such a way that the similarity of graphs from the same class is increased. The matching are computed only once before learning the costs and each edit operation is considered independently from the matching it belongs. From the same authors, in [Neuhaus and Bunke, 2007], the graph matching process is formulated in a stochastic context. A maximum likelihood parameter estimation of the distribution of matching operations is performed. The underlying distortion model is a mixture of multivariate Gaussians. The model is learned using an Expectation Maximization algorithm. The matching costs are adapted to decrease the distances between graphs from the same class, thereby leading to compact graph clusters. The method requires the summation over all possible edit paths between two graphs which is not tractable in practice. But it can be approximated by GED heuristics (A beam search for instance).

4.2.2 Open problems

By reviewing the literature about classification operating in graph space, machine learning can be introduced in two locations:

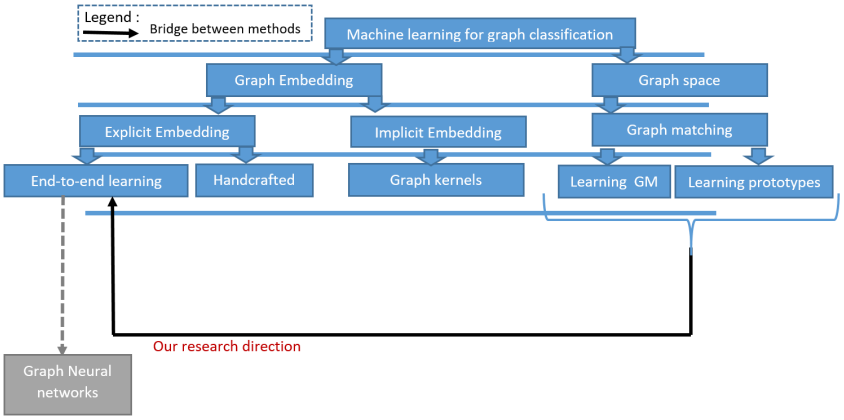


Figure 4.16: Bridging the gap between end-to-end graph embedding and graph space techniques by learning graph matching and prototypes.

1. Learning graph distance
2. Learning graph prototypes

The literature about learning graph distance aimed at learning common weights shared among all the edges' and nodes. Weights are global over a graph pair and not local to a given pair of nodes or edges. To overcome this problem, the discriminative weight formulation used in graph matching context in Equation 4.12 can be of first interest but the learning remains to be designed. Another, room for improvement could be to learn the graph prototypes along with the graph distance. To our knowledge, in the literature, there is no paper about learning graph matching and graph prototypes, at the same time, in the graph space. The most similar idea lies in [Cho et al., 2013] where graph prototypes and graph matching are learned at the same time. However, the approach do not tackle a classification problem and furthermore, the method is not hierarchical. Such that the complicated learning problem must be addressed in a single step. At the opposite, GNNs [Kipf and Welling, 2016, Monti et al., 2016] are organized in a hierarchical manner but graphs are locally projected in the Euclidean space to compute local statistics. This feature vector transformation leads to loss of topological information. Here comes a great objective, to create a GNN operating in the graph space in order to, at once, learn graph matching and graph prototypes. Pictorially, our research direction is showed in Figure 4.16.

The deadlocks to be addressed are :

1. Deadlock 10: Learning graph distance for classification with local parameters for nodes and edges.
2. Deadlock 11: Learning graph matching and graph prototypes in a hierarchical manner.

4.2.3 Contributions

4.2.3.1 Deadlock 10: Learning graph distance for classification with local parameters for nodes and edges

4.2.3.1.1 Motivation Previous works [Neuhaus and Bunke, 2007, Riesen and Bunke, 2009, Cortés and Serratosa, 2015] have considered a global parametrization of the graph distance. Here, we take benefit of the parametrized graph edit distance described in Model 7(PGED) to built a

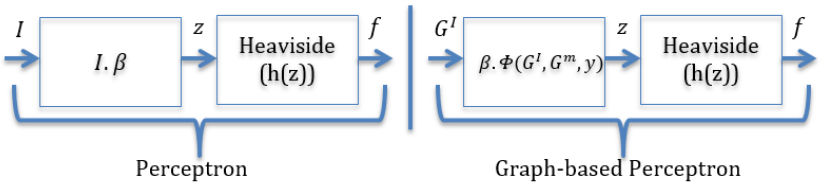


Figure 4.17: Overview of the perceptron and a modified perceptron for graph classification. I in this figure is an input vector.

perceptron neuron operating in graph space for a classification purpose. A weight is associated to each nodes and edges of a graph prototype. The choice of the perceptron may seem dated compared to recent deep neural network architectures [Lecun et al., 1998]. However, the simple perceptron neuron can be stacked to create a multilayer perceptron (MLP). The MLP is a renown algorithm which is still in use in the latest deep learning architectures and is especially involved in the dense layers.

4.2.3.1.2 Details In the context of neural networks, a perceptron is an artificial neuron using the Heaviside step function as the activation function. A global picture of the graph-based perceptron is depicted in Figure 4.17. The conventional perceptron is adapted to graphs thanks to three main features : a) the learning rule for updating the weight vector β, b) the graph matching algorithm for finding y^*, and c) the graph model G_m.

$TrS = \{(G_i, t_i)\}_{i=1}^{M}$ is the training set composed of graphs and their associated class $t_i \in \{0, 1\}$. We first consider a binary or two-class classification problem. The Algorithm 3 is designed to train the perceptron model. The algorithm is operating on the graph set TrS through a change of variable from G to $\Phi(G, G_m, y^*)$. $\Phi(G, G_m, y^*)$ is the joint feature map, the vector of minimal edit costs for each element in the prototype graph G_m. Any type of prototypes could be involved but discriminative graph prototypes could be more suited for the classification task. Then, the weights β are optimized with respect to classification loss through the perceptron algorithm. But β is also implied in finding y^* as $y^* = \underset{y}{arg\,min} \quad d(G, G_m, y, \beta)$. It means modifying β is not only acting on the linear projection of $\Phi(G, G_m, y^*)$ but also on the graph matching operator, which is a non-linear operation. Reinterpreting the minimization problem, it appears that both variables β and y are involved into the minimization of an empirical risk guided by the loss function l:

$$\min_{\beta, y} \sum_{(G_i, t_i) \in TrS} l(G_i, t_i, G_m, y_i, \beta) \tag{4.18}$$

$$l = \frac{1}{2}\left(t_i - heaviside(\beta^T \cdot \Phi(G_i, G_m, y_i^*))\right)^2 \tag{4.19}$$

The $heaviside : \mathbb{R} \to \{0, 1\}$ is a decision function defined as

$$heaviside = \begin{cases} 1 & \text{if } \beta^T \cdot \Phi(G, G_m, y) + b > 0 \\ 0 & \text{otherwise} \end{cases} \text{ with } G_m \text{ a prototype}$$

Such an output is suitable for a binary classification problem.

Now, let us get a closer look to the learning algorithm of the graph-based perceptron for classification. Algorithm 3 is a deterministic algorithm. $\#iter$ is the maximum number of iterations or also called *epochs* in the literature. The parametrized graph matching problem is solved in Line 9. Lines 10 to 14 apply the learning rule defined in Equation 4.20 when the prediction is wrong. To show the time-dependence of β, we use $\beta(i)$ as the weight at time i.

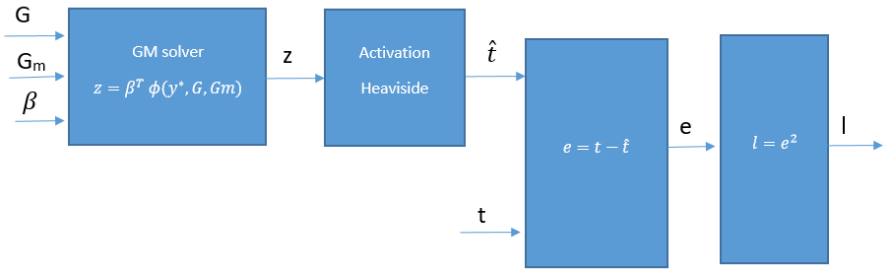


Figure 4.18: Computation graph of our graph-based classifier from the inputs to the loss.

4.2.3.1.2.1 Learning rule. The learning rule aims at modifying β. The weights should be updated in cases of wrong predictions. The correction must take into account the amount and the sign of the committed error.

$$\begin{aligned} \beta(i+1) &\approx \beta(i) - \alpha \frac{\partial l(.)}{\partial \beta(i)} \\ \beta(i+1) &\approx \beta(i) - \alpha(\hat{t} - t)\Phi(G, G_m, y^*) \end{aligned} \tag{4.20}$$

This rule is obtained by deriving the computation graph shown in Figure 4.18 with respect to parameters β.

The calculus of the derivation goes as follows :

$$\frac{\partial l}{\partial e} = e \quad ; \frac{\partial e}{\partial \hat{t}} = -1 \quad ; \frac{\partial \hat{t}}{\partial z} \approx 1 \quad ; \frac{\partial z}{\partial \beta} \approx \phi(y^*) \tag{4.21a}$$

$$\frac{\partial l}{\partial \beta} = \frac{\partial l}{\partial e} . \frac{\partial e}{\partial \hat{t}} . \frac{\partial \hat{t}}{\partial z} \frac{\partial z}{\partial \beta} \tag{4.21b}$$

$$\frac{\partial l}{\partial \beta} \approx -e.\Phi(y^*) \tag{4.21c}$$

$$\frac{\partial l}{\partial \beta} \approx (\hat{t} - t).\Phi(y^*) \tag{4.21d}$$

$$\tag{4.21e}$$

The gradient calculation is not the exact gradient. The Heaviside is no differentiable anywhere. So we approximate it as a linear function to let the gradient flow through the graph matching algorithm. We decided to not propagate gradients inside each iteration of the solver but we do it only once at convergence ($\frac{\partial y^*}{\partial \beta}$). We are aware of that shortcut. The goal was to be faster maybe at the cost of a noisier or unstable gradient. Parameters are updated in the opposite direction of the gradient. The goal is to get closer and closer to the best parameter values β_{min} as shown in Figure 4.11.

Algorithm 3 is a deterministic algorithm whose complexity in terms of calls to the matching solver is $O(\#iter \cdot |TrS|)$ where $\#iter$ is the number of iterations. In addition, the entire test set (TeS) is classified by only $|TeS|$ calls to the graph matching algorithm. This linear complexity makes the decision procedure a fast graph classifier. Classical kNN graph classifiers require generally $|TrS| \times |TeS|$ calls to the graph matching solver.

The methods can also be seen as learning a discriminative distance function between a training set and a graph prototype (G_m).

Algorithm 3 Training the graph-based perceptron for classification.

INPUT: $TrS = \{(G,t)\}$ and G_m
INPUT: $\#iter$ is the maximum number of iterations
INPUT: α learning rate
OUTPUT: Learned β. A weight vector
Init: $\beta \leftarrow \mathbf{1}$ and $iter \leftarrow 0$
while iter< $\#iter$ **do**
  i $\leftarrow 0$
  for $(G,t) \in TrS$ **do**
    $y^* \leftarrow \underset{y \in \Gamma(G,G_m)}{\arg\min} \quad \beta(i)^T \cdot \Phi(G, G_m, y)$ `// Solve Model 7`
    $z \leftarrow \beta(i)^T \cdot \Phi(G, G_m, y^*)$
    $\hat{t} \leftarrow heaviside(z)$
    if $\hat{t} \neq t$ **then**
      $\beta(i+1) \leftarrow \beta(i) - \alpha(t - \hat{t})\Phi(G, G_m, y^*)$
    end if
    i $\leftarrow$ i +1
  end for
  iter $\leftarrow$ iter +1
end while

The perceptron provides a natural extension to the multiclass problem. The Heaviside function could be replaced by a $Relu(z) = \max(0, z)$. Then, instead of having only one neuron with binary output, we could have m neurons leading to multiclass classification. A set of functions $f(G,t)$ map each possible input/output pair to a real value that represents the fitness of the pair (G,t). The resulting score is used to choose among many possible outputs: $\hat{t} = \text{argmin}_t\, f(G,t)$.

One drawback of the method is the *a priori* choice of the prototype graph (G_m). We have seen in the state of the art that computing a prototype (G_m) is dependent on a distance function. The distance is exactly what our method aims to learn. So we face a chicken-egg problem. Consequently, the choice of G_m is made based on a arbitrary selected distance function. In other words, the prototype graph computation requires pre-defined weights. To overcome this issue, a looping process with 2 phases could be envisaged but is not applied here : 1) compute the prototype graph, 2) learn the distance parameters and loop to step 1).

4.2.3.1.2.2 Experiments The proposed approach stands and falls with the answer to the overall question, whether or not we are able to outperform traditional pattern recognition systems that are directly applied in the graph domain. That is, the essential and sole quality criterion to be applied to the novel approach is the degree of improvement in the recognition accuracy. More pessimistically one might ask, is there any improvement at all?

Algorithm 3 requires a GED heuristic. The method called BP [Riesen and Bunke, 2009] has been chosen because it is fast but another could be chosen. Prototypes are computed according to the Definition 13 of the set median graphs. Predefined β parameters are required to compute the median graphs of the G-M-Perceptron method. A vanilla plain solution was adopted with $\beta = \mathbf{1}$. Let us recall that $\beta = \mathbf{1}$ is the initialization value of our algorithms.

The datasets are described in Table 4.6. These databases are representative of a wide range of learning problems that occur in Computer Vision. Matching functions d_V and d_E were taken from [Riesen and Bunke, 2010b] and [Moreno-García et al., 2016].

A commonly used approach in pattern classification is based on nearest-neighbor classification. That is, an unknown object is assigned the class of its closest known element, or nearest neighbor

(1NN). Two versions were utilized in the tests. R-1NN [Riesen and Bunke, 2009] and C-1NN [Cortés and Serratosa, 2015] where the values of $K_V = c(i \rightarrow \epsilon) = c(\epsilon \rightarrow k)$ and $K_E = c(ij \rightarrow \epsilon) = c(\epsilon \rightarrow kl)$ were borrowed from [Riesen and Bunke, 2010b] and [Moreno-García et al., 2016], respectively. K_N corresponds to the cost of a node deletion or insertion, and K_E corresponds to the cost of an edge deletion or insertion. The aforementioned methods hold a meta parameter $\alpha \in [0, 1]$ which corresponds to the weighting parameter that controls whether the cost on the nodes or on the edges is more important. In Table 4.7, the best values of parameters (α, K_N, K_E) are summarized.

In Figure 4.19, the impact of the learning rate is depicted for the median models. A very high learning rate ($\alpha = 0.1$) leads to poor results. The search space exploration is too fast and saddle points are missed. A high learning rate ($\alpha = 0.01$) leads to unstable results with many oscillations while a low learning rate implies a slow but smooth convergence. A trade-off can be achieved with an intermediate value ($\alpha = 0.001$). For the rest of the experiments, $\alpha = 0.001$ was chosen and the number of iterations was set to 300.

To summarize the results of theses experiments, the classification rates (η) during the training and the test phases are reported in Table 4.9 along with the time, in **milliseconds**, for classifying all test instances. Table 4.9, the classification rates obtained during the learning phase are tabulated (column η_{TrS}). The learning ability is demonstrated on all data sets. The classification rate is always higher on the training set than on the test set except on the CMU and Fingerprint databases where the number of classes is small. The gap between the training and test recognition rates is 3% on average. This result demonstrates the good generalization ability of our algorithms.

Classification results on 7 publicly available datasets demonstrated a large speed-up during the test phase (60 times faster in average) with a loss of accuracy of 6% on average compared to a 1NN classifier based on an optimized graph distance.

Database	size (TrS,TeS)	♯classes	node labels	edge labels	$\overline{\lvert V \rvert}$	$\overline{\lvert E \rvert}$	max $\lvert V \rvert$	max $\lvert E \rvert$	balanced
LETTER-HIGH	(750,750)	15	x,y	none	4.7	4.5	9	9	Y
LETTER-MED	(750,750)	15	x,y	none	4.7	3.2	9	9	Y
LETTER-LOW	(750,750)	15	x,y	none	4.7	3.1	9	9	Y
CMU	(71,70)	2	Shape	none	30	154.4	30	158	Y
GREC	(286,528)	22	x,y	Line types	11.5	12.2	25	30	Y
Fingerprint	(378,1533)	4	none	angle	5.42	4.42	26	24	N
COIL-DEL	(2400,1000)	100	x,y	none	21.5	54.2	77	222	Y

Table 4.6: Summary of graph data set characteristics.

4.2.3.2 Deadlock 11: Learning graph matching and graph prototypes in a hierarchical manner

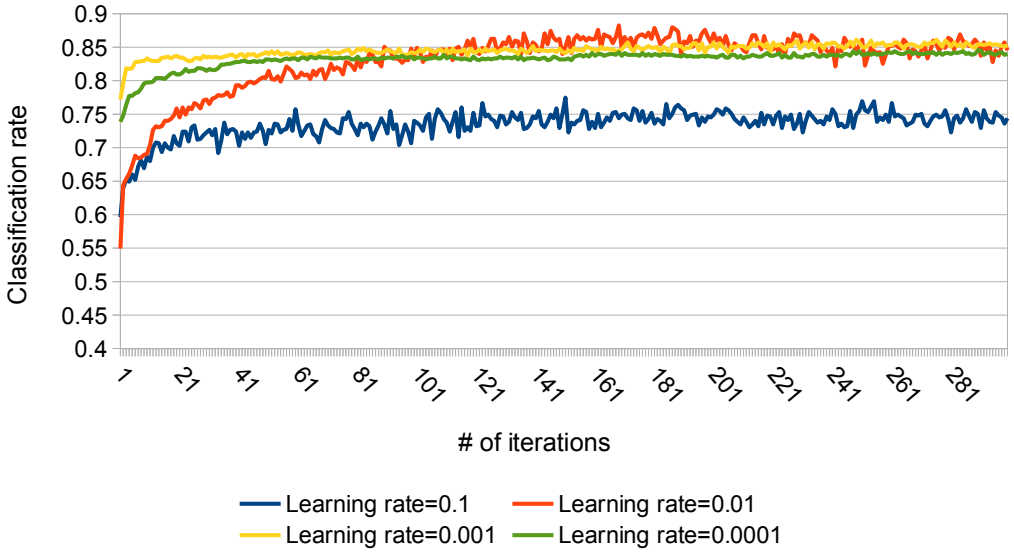
4.2.3.2.1 Motivation A first drawback of the graph-based perceptron is that it does not learn the prototype graph. The latter must be computed before. Second, the graph-based perceptron

Database	R-1-NN				C-1-NN			
	α	K_n	K_e	From	α	K_n	K_e	From
LETTER-LOW	0.3	0.1	0.25	Paper I	0.5	1	1	Paper II
LETTER-MED	0.7	01.9	0.75	Paper I	0.5	1	1	Paper II
LETTER-HIGH	0.9	1.7	0.75	Paper I	0.5	1	1	Paper II
CMU	NA	NA	NA	NA	0.5	1000	1	Paper II
Fingerprint	0.75	0.7	0.5	Paper I	NA	NA	*NA*	NA
COIL-DEL	NA	NA	NA	NA	NA	NA	*NA*	NA

Table 4.7: Best parameters according to learning strategies R-1-NN and C-1-NN taken from Paper I: [Riesen and Bunke, 2010b] and from Paper II: [Moreno-García et al., 2016]

Table 4.8: Taxonomy of the compared methods

Classifier	Parameter learning	Model graph	Method's Name
1NN	R [Riesen and Bunke, 2009]	M	R-M-1NN
		TrS	R-1NN
	C [Cortés and Serratosa, 2015]	M	C-M-1NN
		TrS	C-1NN
Perceptron	Gradient descent	M	G-M-Perceptron



(a) Median graph model

Figure 4.19: Letter-HIGH : Impact of the learning rate on the convergence.

output a scalar value. By replacing the heaviside function by a $Relu(z) = \max(0, z)$, the graph-based perceptron could be plugged/stacked with standard MLP units. Therefore, the the graph-based perceptron would only play the role of a first input layer absorbing the graph data structure. Here comes the need to have graph-based neurons at each layer of a deep learning architecture.

GNN that are reviewed, in Appendix C, rely mainly on a vector space definition of locality in graphs, we propose to stay in graph space by using a convolution operator based on graph matching. The parametrized convolution operator based on graph matching is depicted in Figure 4.21. A convolution filter is an attributed graph. The result of our convolution operator on an input graph is, for a given node i, the matching similarity between $N(i)$ the neighbourhood of i and the graph filter. The intuition is that the local response of a convolution filter on euclidean data is analog to graph similarity between a node neighbourhood and the filter graph. A filter graph can be seen as a prototype dedicated to react to a pattern within the input graph. To illustrate this principle, we propose some toy examples on images represented by regular grids. In Figure 4.20, the results of convolutions is displayed. The filter graph is a simple graph with 2 nodes and one edge. Nodes are labelled with values -1 and 1 respectively. By analogy with image processing, this filter graph can be seen as a simple gradient kernel. The convolution operator endowed with this specific filter will find the maximum contrast (pairwise pixel differences) inside

Table 4.9: Classification results. The best classification rates are marked in blue while the best processing times are in red. "Best" means an improvement over other methods statistically significant according to a z-test with a confidence level of 70%

	G-M-Perceptron			R-1NN		C-1NN		R-M-1NN		C-M-1NN	
Database	η_{TrS}	η_{TeS}	Time	η_{TeS}	Time	η_{TeS}	Time	η_{TeS}	Time	η_{TeS}	Time
LETTER-LOW	1	0.98	1436	0.99	69700	0.96	71869	0.97	1341	0.98	1341
LETTER-MED	0.90	0.87	1404	0.92	74506	0.93	72150	0.86	1388	0.81	1310
LETTER-HIGH	0.86	0.81	1669	0.83	86377	0.84	84911	0.82	1731	0.71	1498
CMU	1	0.99	11029	NA	NA	0.99	427955	NA	NA	0.99	10780
GREC	0.84	0.75	14370	0.98	199087	NA	NA	0.96	14726	NA	NA
Fingerprint	0.74	0.76	1576	0.58	260816	NA	NA	0.73	2138	NA	NA
Coil-DEL	0.52	0.52	471575	NA	NA	NA	NA	NA	NA	NA	NA

a given neighbourhood $N(i)$. The key difference, with the image domain, is that the filter does not need to be "spatialised". There is no need to apply the filter vertically (y axis) and horizontally (x axis). Therefore the convolution is rotation invariant.

Several graph filters can be added to compose a convolution layer as shown in Figure 4.22. Finally, in the GNN review (Appendix C), we have seen that edge attributes were not fully handled. They were either restricted to be a scalar value or merely aggregated with the node features.

GNN described in the literature outputs node embeddings but never edge embedddings. When describing the convolution with an error-tolerant graph matching operator, this problem is solved. Graph matching can manage complex edge attributes. The outputs of our graph matching based neural network are node and edge embedddings.

4.2.3.2.2 Details Now, that we have drawn a global picture of our GNN based on graph matching. Let us define in greater detail the framework that is at its early stage of development. To define our convolution operator, we must define the graph matching function that will be pointwisely used.

Definition 18. *Graph Matching Similarity*

Let G_1 and G_2 be attributed graphs: $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (V_2, E_2, \mu_2, \zeta_2)$. n_1 and n_2 are the size of the sets V_1 and V_2, respectively.

$$GMS(G_1, G_2) = \max_{y} \quad s(G_1, G_2, y), \tag{4.22a}$$

$$\text{subject to} \quad y \in \{0,1\}^{n_1 n_2} \tag{4.22b}$$

$$\sum_{i=1}^{n_1} y_{i,k} \leq 1 \quad \forall k \in [1, \cdots, n_2] \tag{4.22c}$$

$$\sum_{k=1}^{n_2} y_{i,k} \leq 1 \quad \forall i \in [1, \cdots, n_1] \tag{4.22d}$$

$$\tag{4.22e}$$

The choice of this graph matching problem is driven by the analogy with the CNN. Where a convolution response is maximum when the filter fits well to the input signal.

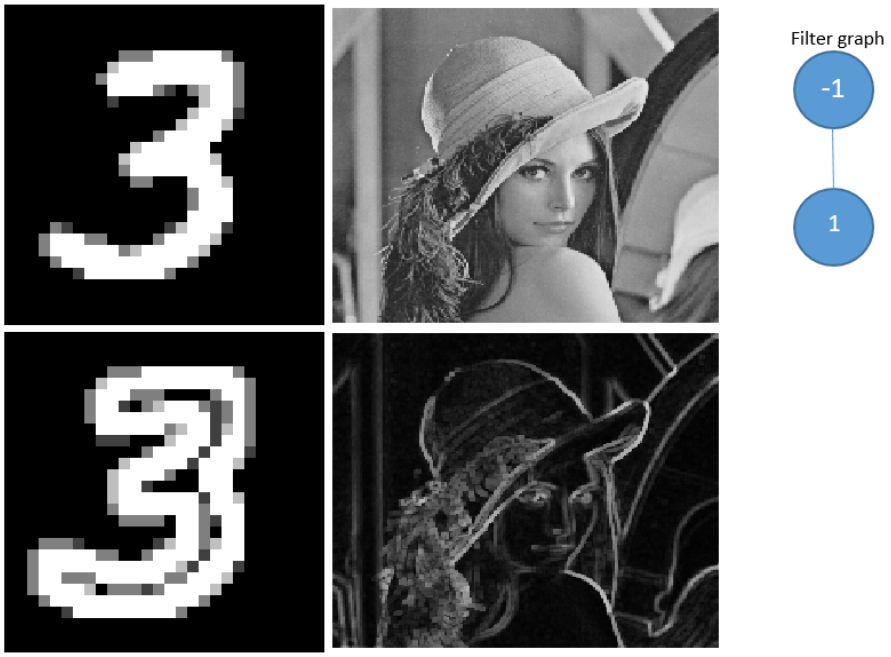


Figure 4.20: Top: Original images. Bottom: Results of a filter graph convolves on regular grids of images (8-connexity). The neighbourhood is a one hop neighbourhood.

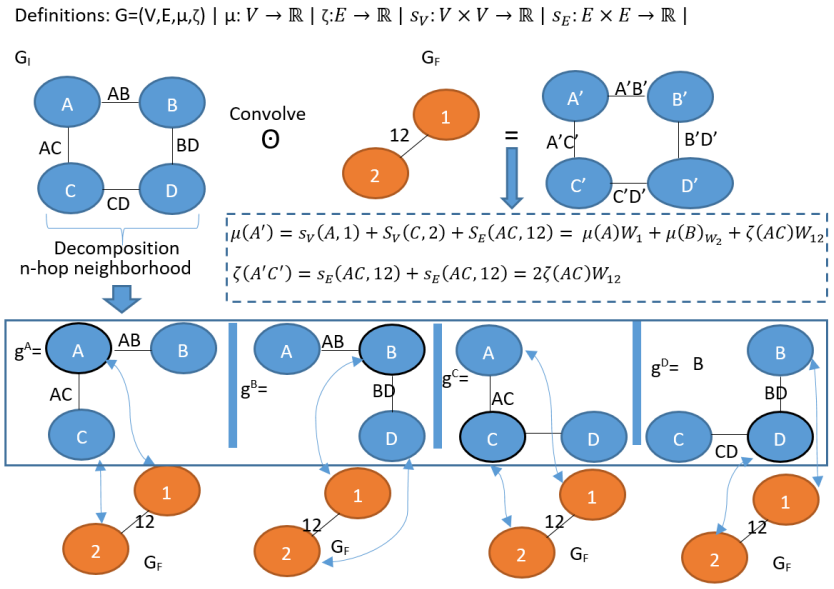


Figure 4.21: Two graphs are convolved. G_I is the input graph and G_F is the filter graph. Attributes of G_F are parameters W. Convolution is based on matching G_F at different locations of the input graph.

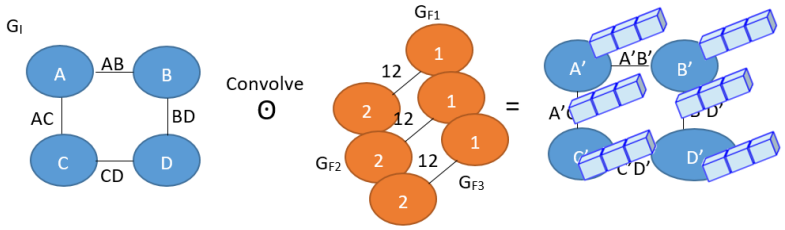


Figure 4.22: Graph matching based convolution layer.

The similarity function s is defined as follows:

$$s(G_1, G_2, y) = \sum_{y_{ik}=1} s_V(i,k) + \sum_{y_{ik}=1} \sum_{y_{jl}=1} s_E(ij,kl) \tag{4.23a}$$

$$s_V(i,k) = \mu(i).\mu(k) \tag{4.23b}$$

$$s_E(ij,kl) = \zeta(ij).\zeta(kl) \tag{4.23c}$$

Similarity between nodes and edges are defined as dot products.

Now that our matching operator is formulated, we can apply it over an input graph to compute the result of convolution.

Let G_I and G_F be attributed graphs: $G_I = (V_I, E_I, \mu_I, \zeta_I)$ and $G_F = (V_F, E_F, \mu_F, \zeta_F)$. G_I and G_F are respectively referred to as the input graph and the filter graph.

Definition 19. *Graph convolution operator* $\odot$

$$G_I \odot G_F = (V_I, E_I, \mu, \zeta) \tag{4.24a}$$

$$\textit{with} \quad \mu : V_I \to \mathbb{R} \quad \textit{such that} \quad \mu(i) = GMS(g_I^i, G_F) \, \forall \, i \in V_I \tag{4.24b}$$

$$\zeta : E_I \to \mathbb{R} \quad \textit{such that} \quad \zeta(ij) = score(ij, G_I, G_F) \, \forall \, ij \in E_I \tag{4.24c}$$

.

g_I^i is defining the neighbourhood (which is a subgraph) for vertex i in G_I.

The results of the convolution operator is a graph with the same topology/structure than G_I but with different attributes on edges and nodes.

4.2.3.2.2.1 Edge attribute in convolved graph score is a function mapping an edge to its matching score in the found GMS. The problem is that it might be assigned multiple times:

$$\text{let} \quad \mathcal{P}_{ij} = \{ij \in g_I^i | g_I^i \quad \forall i \in V_I\} \quad \forall ij \in E_I \tag{4.25}$$

$\mathcal{P}_{ij}$ potentially contains more than one element. Therefore, a score can be defined as follows:

$$\text{score}(ij, G_I, G_F) = \theta\left(\{s_E(ij,kl).y_{ij,kl} \quad \forall kl \in E_F \quad \forall g_I \in \mathcal{P}_{ij}\}\right) \tag{4.26a}$$

$$\text{with} \quad \theta : \text{some statistical estimator (max or avg)} \tag{4.26b}$$

Now that the convolution operator is defined, it is possible to use it as a base to build a convolution layer. This layer can be included in a graph neural network.

Definition 20. *Filter graph*
A filter graph is an attributed graph G_F^W. Every attribute function is parametrized with respect to weight vector $W \in \mathbb{R}^{|V|+|E|+2}$. The extra parameters (noted W^{b1} and W^{b2}) correspond to the biases w.r.t vertex and edges outputs.

$$G_F^W = (V_F, E_F, \mu_F^W, \zeta_F^W) \tag{4.27a}$$

$$with \quad \mu_F^W(k) = W_k \tag{4.27b}$$

$$\zeta_F^W(kl) = W_{kl} \tag{4.27c}$$

Definition 21. *Graph convolution filter*
A graph convolution filter is a convolution operation based on a filter graph. The output of graph convolution filter $convfilter : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ is defined as follows:

$$convfilter(G_I, G_F^W) = G_I \odot G_F^W + W^{b\cdot} \tag{4.28a}$$

$$\tag{4.28b}$$

Definition 22. *Graph convolution layer*
A convolution layer is a set of filter graphs $\{G_F^k\}_{k=1}^k$ endowed with a convolution operator and applied on a same input graph G_I.

The output function of the layer is a graph with same topology as G_I but with attributes as vectors composed by attributes of every filters outputs.

Now let us how this layer can be integrated into a deep learning architecture.

4.2.3.2.2.2 Architecture GNN based on our graph convolution layer can perform both graph classification or node classification. Graph convolution layers can be stacked and combined with conventional layers or activation functions like RelU or Softmax. Only the pooling layer should be adapted to graphs. The graph pooling goals are: a) pool similar local features (max pooling or average pooling) and b) series of pooling layers create invariance to global geometric deformations (translation invariance for instance). The main challenge is to design a multi-scale coarsening algorithm that preserves non-linear graph structures. Graph coarsening decomposes G_I into smaller meaningful clusters. This problem is combinatorial and is $\mathcal{NP}$-hard. In the literature, some graph clustering algorithm have been favored such as Graclus or the Louvain method [Dhillon et al., 2007, Blondel et al., 2008].

4.2.3.2.2.3 Hyperparameters Like any traditional convolution layer, hyperparameters of our graph convolution layer are the number of filter graphs and their size. Moreover, another hyperparameter is introduced. The number of hops (n-hops) that defines a node neighborhood. This hyperparameter $n - hops$ is involved in the creation of the subgraph g_I^k rooted in node k. In the experiment, the $n - hops$ parameter is set to one. The choice of the graph maching solver is also an important element. In the experiment, the BP algorithm was chosen [Riesen and Bunke, 2009]. This choices are made arbitrary to establish preliminary results. A deeper study about these hyper parameters should be carried out in a near future.

4.2.3.2.2.4 Experiments We applied the proposed method on a classical task of hand-written digit classification in the MNIST dataset. While almost trivial by todays standards, we nevertheless use this example to obtain preliminary results of our approach. Our experimental setup followed [Monti et al., 2016]. The 28×28 images were represented as graphs, where vertices

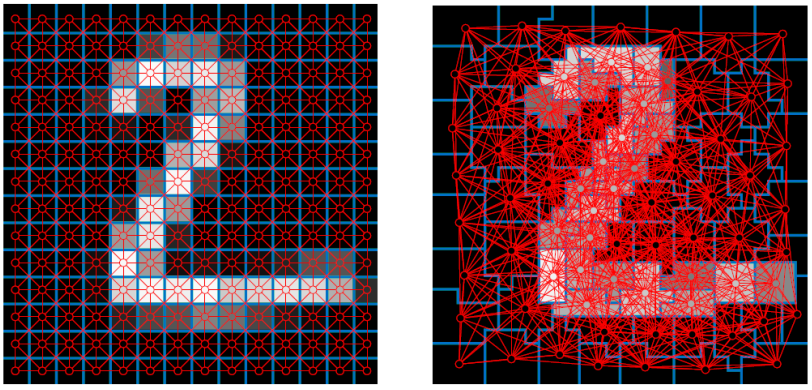


Figure 4.23: MNIST digits classiffication (Regular grid, Superpixels) (Taken from CVPR 2017 tutorial http://geometricdeeplearning.com/).

Table 4.10: Recognition rates on MNIST 2class

Representation	**Dataset**	**CNN**		**MoNet**		**Ours**	
		Valid	Test	Valid	Test	Valid	Test
$\frac{1}{4}$ grid	MNIST	100 %	99.88 %	97.56 %	99.40 %	99.51 %	97.76 %
75 superpixels	MNIST			94.13 %	92.70 %	94.13 %	89.53 %

correspond to (super)pixels and edges represent their spatial relations. We considered two constructions: all images represented on the same graph (regular grid) and each image represented as a different graph (see Figure 4.23 left and right, respectively). Furthermore, the grids contain 196 vertices and the superpixel-based graphs contain 75 vertices.

Three methods were compared: classical CNN LeNet5 architecture [Lecun et al., 1998] (containing two convolutional, two max pooling, and one fully-connected layer, applied on regular grids only), MoNet [Monti et al., 2016] that is a recent and effective GNN and our proposal. We used a standard splitting of the MNIST dataset into training, testing, and validation sets of sizes 55K, 10K, and 5K images, respectively. LeNet used 2×2 max-pooling; and for MoNet and our method, we used three convolutional layers, interleaved with pooling layers based on the Graclus method [Dhillon et al., 2007] to coarsen the graph by a factor of four.

Training was done with 350K iterations of Adam optimizer, initial learning rate 10^{-4} regularization factor 10^{-4} dropout probability 0.5, and batch size of 10.

Results on MNIST 2class are reported in Table 4.10.

The overall tendency of results show a clear advantage for the CNN when dealing with grid graphs. MoNet is the best when dealing with the irregular graphs. Nevertheless, our method is just at the early stage of development. The $n-hops$ parameter was set to one. Higher value could be a plus to capture more structural information. In addition, filter sizes were set up like in the LeNET5 network. The filter size is 5 nodes so graph prototypes are rather small. Larger graph prototypes could capture more structural information too. Better graph matching solver could be investigated. Especially, the GM solver adopted in [Zanfir and Sminchisescu, 2018] could be a good choice. It seems to be faster and easier to optimize by back propagation.

4.2.4 Summary

In the PhD of Maxime Martineau, the research direction of bridging the gap between graph space and explicit embedding was investigated. A first step in this direction was to propose a graph-based perceptron for learning discriminative graph matching in a classification context [Martineau et al., 2018, in press]. Graph matching was parametrized to build a weighted formulation. This weighted formulation was used to define a perceptron classifier, in which each neuron is composed of a prototype graph and a vector of parameters. Each weight is associated with a graph component of the prototype graph. Weights are learned using the gradient descent algorithm. A main drawback is that the prototype graph is not learned and the method is not hierarchical. So secondly, a convolution operator based on graph matching was proposed and integrated into a GNN architecture. Convolution filters play the role of prototypes graphs to be learned. Such a GNN model offer an elegant solution to output both node and edge embeddings.

Chapter 5

Conclusion and perspectives

5.1 Conclusion

This manuscript addresses the issue of matching and classifying graphs. In structural pattern recognition based on graphs, the idea is to transform patterns into graphs and then perform the analysis and classification of patterns in the domain of graphs.

Graphs are very flexible computer data structures that allow a very rich and very detailed description of a very wide range of objects, ranging from chemical molecules to images, via social networks. Paradoxically, despite the important power of representation of graphs, the Machine Learning and Operational Research communities have not mixed closely to develop powerful algorithms for analyzing data represented by graphs.

My research activities are focused around pattern recognition using structural methods. Two axes are developed during my research: discrete optimization and learning in the space of graphs. The applications are matching and classifying graphs.

5.1.1 Discrete optimization for graph matching and graph classification

In this section are discussed the issues related to the calculation of graph matching as well as the classification of graphs.

5.1.1.1 Graph comparison

Many applications, such as information retrieval or classification, require measuring the distance or similarity between two graphs, ie, matching - the vertices of the graphs to identify their common points and their differences. A first objective was to define new mathematical models to represent the problem of the graph edit distance. In a collaboration with LITIS Lab (in Rouen) [Lerouge et al., 2017, 2016] and also the PhD of Mostafa Darwiche [Darwiche et al., 2018, in pressa], three models based on integer linear programming were developed. This formalism makes possible to benefit from efficient solving methods that can be easily exploited thanks to solvers. A solver is a computer software capable of solving mathematical equations or logic problems.

The graph edit distance is an $\mathcal{NP}$-hard optimization problem. Its solution time increases exponentially according to the number of nodes of the two graphs. Therefore, two challenges arise for this type of problem. First, the development of exact methods to obtain the optimal solution of the problem quickly. Exact solving is not always possible in practice because of the combinatorial explosion caused by the complexity of the problem. From this observation arose the

second challenge, the design of heuristic methods able to quickly provide a sub-optimal solution of quality.

We have proposed two exact methods to calculate the optimal solution of the optimization problem. In Zeina Abu-aisheh thesis [Abu-Aisheh et al., 2015b], a branch and bound method was proposed. An evaluation of all possible solutions is performed without explicitly listing them. Partial solutions are removed using the lower and upper bounds. In the PhD of Mostafa Darwiche, the use of a mathematical solver solved the three formulations based on the integer linear programming. The couple integer linear programming and mathematical solver allowed to obtain the best results. To date, the LITIS and LIFAT have among the best exact methods for solving the graph edit distance problem. Since exact solving is not always possible in practice, we are interested in heuristic methods. In my thesis [Raveaux, 2010], I explored the possibility of simplifying the initial problem to transform it into a linear subgraph assignment problem whose solving is in polynomial time and no longer in exponential time. Of course, this time reduction is not without consequence on the quality of the solution obtained. There is no free lunch [1]. In Zeina Abu-aisheh's thesis [Abu-Aisheh, 2016] and in collaboration with the LITIS [Lerouge et al., 2017], heuristics are obtained simply by limiting execution time of exact methods. In doing so, it is easy to meet the time constraints of certain applications but no information on the quality of the returned solution is taken into account to stop the method. This drawback is raised in the PhD of Mostafa Darwiche [Darwiche et al., 2018, in pressa], two local searches, in the sense of a neighborhood operator in the solution space, based on the linear programming and a mathematical solver have been proposed. These methods explore the solution space locally around a neighborhood and stop if no improved solution is found. These heuristics are among the most accurate of the literature. In addition, mathematical solvers, such as IBM CPLEX for example, are constantly evolving and become more and more efficient from year to year. This suggests that the approaches developed by LIFAT will become even more effective in the future.

Taking a step back on the methods helped to bring closer the notions of exact method and heuristic by proposing the methods called anytime [Abu-Aisheh et al., 2017a]. This type of method is capable of delivering a first feasible solution very quickly and then gradually improving it to converge towards an optimal solution. Whenever an improved solution is found, it is made available for the final application that uses the anytime method as a service of solutions. This way of understanding the problem makes the anytime method very flexible and applicable when the time constraints of the final application are not known in advance.

Finally, a solid methodological benchmark consisting of graphs and metrics has been proposed for the performance evaluation [Abu-Aisheh et al., 2015a] of the graph edit distance. From this approach was born a competition on this problem within the framework of the international conference in pattern recognition ICPR 2016 in collaboration with colleagues of the GREYC laboratory in Caen.

5.1.1.2 Graph distance based classification

This part deals with the problem of supervised classification of graphs. In many applications, it is necessary to assign a class (category) to an unknown graph (G). This classification step is based on a set of graphs whose class is known. This set of graphs is called the training set. One of the significant limitations of (dis)similarity based algorithms is that the kernel or distance functions must be evaluated for all possible pairs G and G_i of training data. It can be computationally infeasible during training and it can lead to excessive computation times when making predictions for new graphs. Three different methodological angles correct these defects:

- 1) The use of a fast heuristic to compute the graph edit distance. This solution has been

[1] Economist Milton Friedman's sentence

used in the PhD of Zeina Abu-aisheh [Abu-Aisheh et al., 2018, in press] as well as in my thesis [Raveaux et al., 2010, 2013a].

- 2) It is also possible to "reduce" the training set by selecting or generating representatives from the initial database. In my thesis and in a collaboration with the LITIS, I was interested in calculating median graphs (modeling a class) and prototype graphs (discriminant) [Raveaux et al., 2011]. These aspects of selection or generation of prototypes fall under the learning section that will be developed after.

- 3) Finally, the last angle of attack is to model the problem of computing classification problem as a discrete optimization problem and to solve it heuristically. This original methodology was validated in a work with Zeina Abu-aisheh [Abu-Aisheh et al., 2018, in press]. The graph edit distance problem was generalized to comparing a graph G_1 with an entire training set. This approach enables a branch and bound procedure which eliminates comparisons of non-promising graphs thanks to the upper and lower bounds calculated on the classification problem. The result is an optimized exploration of the comparison tree.

5.1.2 Graph matching and graph classification in graph space

In this section is discussed the development and use of learning methods to solve problems of graph matching and graph classification.

5.1.2.1 Graph matching

In Maxime Martineau's thesis, the problem of the graph edit distance has been parametrized to be suitable for learning [Raveaux et al., 2017]. The parameters w weight the cost functions between two nodes or two edges so the problem of matching "node to node" or "edge to edge" is dependent on continuous (real) parameters. The objective of the learning algorithm is to find the values of the parameters that minimize a criterion defined on the training data (empirical risk). An example of a criterion to be minimized is the sum of squared errors between suboptimal and exact matchings. The goal is to have an heuristic as accurate as an exact method. This problem is similar to a structured regression problem. An effective tool for solving a regression problem is a model based on a neural network. Such a model is trained by the gradient descent method. In this context, learning consists of finding the values of the parameters w. The parameterized editing distance can be seen as a particular layer called a combinational layer that is integrated into a neural network. This combinatorial layer requires one hyper-parameter: a prototype graph that must be fixed during the learning phase. The disadvantage of this method is that it does not generalize the learned parameters to several prototype graphs. The parameters w are therefore associated with a particular prototype graph.

5.1.2.2 Graph classification

The previous approach has been adapted to classify graphs [Martineau et al., 2018, in press]. The criterion to be minimized is a misclassification rate. In the context of a two-class problem {0,1 }, the output of the combinatorial layer feeds an function ($H : \mathbb{R} \rightarrow \{0, 1\}$) to predict the class. The inputs of the combinatorial layer remain similar to the previous approach: a fixed prototype graph during the learning phase. In my thesis and in collaboration with the LITIS [Raveaux et al., 2011], we defined four types of prototype graph: 1) the set graphs belonging to the training set, 2) the generalized prototypes that do not to belong to the training set, generalized graphs are synthetic graphs resulting from a generation process. Each of these two large families is divided into two parts: 3) the median graphs are constructed with respect to a single class of the classification

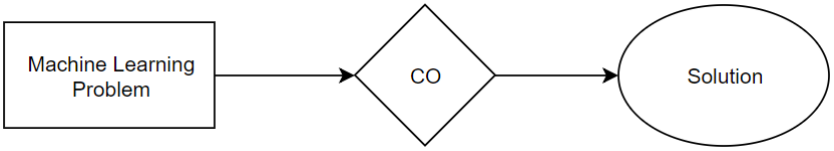


Figure 5.1: The combinatorial optimization (CO) solves a machine learning problem.

problem without taking into consideration the other classes. In contrast, 4) the discriminant graphs are constructed taking into account the entire training set in order to minimize a misclassification rate. In [Martineau et al., 2018, in press], the prototype graph is chosen *a priori* by calculating a set median graph. In Maxime Martineau's thesis, the previous perceptron architecture has been extended to the concept of convolutional neural network (GCNN). This strategy allow to get rid off *a priori* prototypes for the benefit of discriminating graphs determined during the learning phase. All of this work constitutes a solid foundation for extending deep learning networks in graph space.

5.1.3 Interplay between machine learning and combinatorial optimization

As we have seen in this manuscript, machine learning and combinatorial optimization are closely coupled. Some learning problems can be formulated as combinatorial optimization problems as shown in Figure 5.1. It is the case of learning prototype graphs for instance. Here are some other examples that fall in this category:

- k-medians clustering.
- The graph partition problems (Graph cut).
- The MAP-inference problem of a discrete variable CRF.

Combinatorial optimization is then very important for the machine learning community. Better solving methods can lead to better learning scheme. The reverse is also true. Heuristic methods can be improved when they integrate machine learning methods. This idea is presented in Figure 5.2. It is the case of learning schemes for structured outputs that generalize traditional machine learning approaches to structured outputs:

- SVMs → Structured SVM [Tsochantaridis et al., 2004]
- Logistic Regression → Conditional Random Fields [Lafferty et al., 2001]
- Perceptron → Structured Perceptron [Collins, 2002]

All these algorithms learn parameters of a combinatorial optimization problem in order to guide its solving. Studying the benefits of embedding machine learning algorithms into discrete optimization methods is important to create new efficient optimization approaches. Such learning methods is of interest to the discrete optimization community and machine learning community.

Finally, ML can be used to directly output solutions of combinatorial optimization problems [Nowak et al., 2017, Scarselli et al., 2009]. This kind of approach is depicted in Figure 5.3. Such an approach can be guided by the nature of the application that requires to output solutions in real time. ML turns out to be suitable for obtaining accurate solutions in short computing times because some of the complexity is addressed offline.

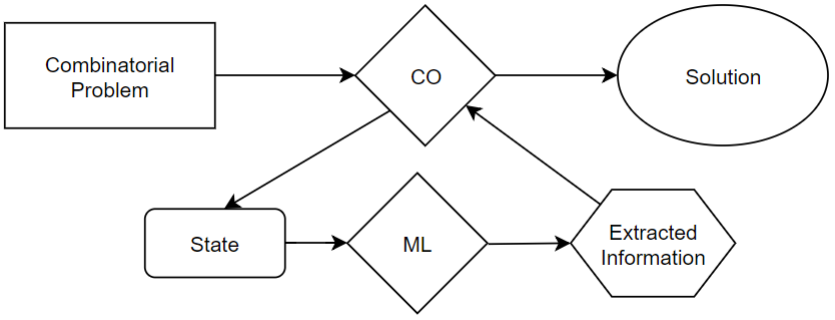


Figure 5.2: The combinatorial optimization (CO) algorithm repeatedly queries the same ML model to make decisions. The ML model takes as input the current state of the combinatorial algorithm.

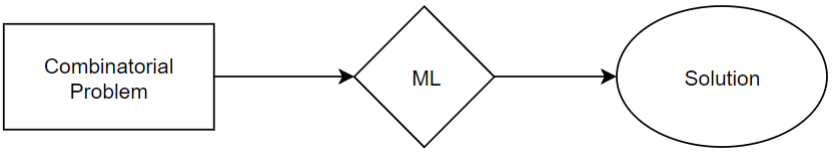


Figure 5.3: Machine learning acts alone to provide a solution to the problem.

5.2 Short term perspectives

This section is concerned by short term perspectives that could lead to PhD subjects. First, we draw our attention on new ideas in the field of learning-free graph matching methods.

5.2.1 Learning-free graph matching problems

5.2.1.1 Higher order graph matching

The graph matching problems consider unary and binary relations so that the matching costs depend on very local information. In pattern recognition, it could be interesting to extend the context to capture a more global information. The matching cost could carry more semantic because the subgraphs to be matched represent larger parts of an object. This paradigm is depicted in Figure 5.4 and modeled by Equation 5.1. Note that only the 3rd order extension is presented for clarity reasons but it could be extended to higher orders.

$$d(G_1, G_2, y) = \sum_{\substack{i \in V_1 \\ k \in V_2}} c(i,k).y_{ik} + \sum_{\substack{ij \in V_1 \times V_1 \\ kl \in V_2 \times V_2}} c(ij,kl).y_{ik,kl} + \sum_{\substack{hij \in V_1^3 \\ mkl \in V_2^3}} c(hij,mkl).y_{hik,mkl} \tag{5.1}$$

To my knowledge, the solving of higher order graph matching for pattern recognition has not been (well) investigated in the literature.

5.2.1.2 A "robust" graph matching model

In the same vein, new models could be investigated. Graph matching problems aims at minimizing a sum of costs. In a pattern recognition context, it could be great to integrate an additional criterion to make the matching more robust to large distortions or outliers. In this objective, the

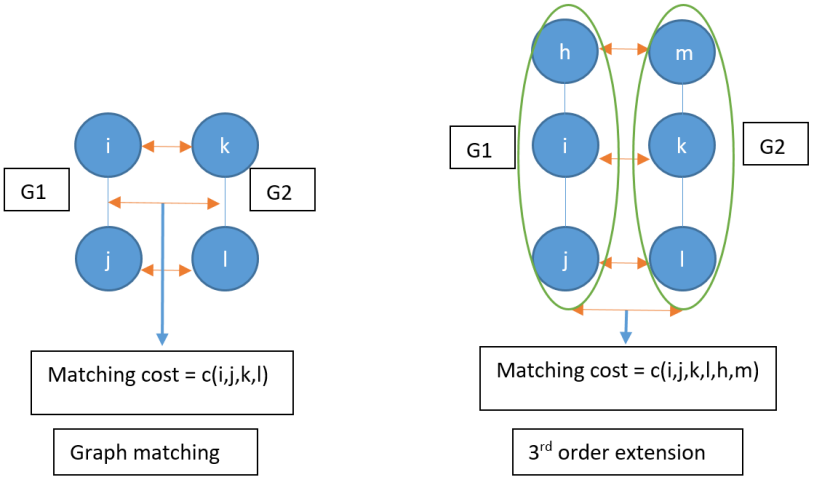


Figure 5.4: Extension of the graph matching problem to a higher order

addition of a penalization term, such that the maximum distortion is small, is of first interest. This intuition is modeled by Equation 5.2.

$$d(G_1, G_2) = \min_{\lambda \in \Gamma(G_1, G_2)} \left[\frac{1}{|\lambda|} \sum_{o_i \in \lambda} c(o_i) + \max_{o_i \in \lambda} c(o_i) \right] \tag{5.2}$$

5.2.1.3 Multivalent matching

In graph matching, the constraints implies that each node can be assigned at most once. In many real-world applications, comparing patterns described at different granularity levels is of great interest. For instance, in the field of image analysis, an over-segmentation of some images might occur whereas an under-estimation occurs in some other images resulting in allowing several regions of one image to be correspondent, or related to, a single region of another image. Based on this fact, multivalent matching problem emerged to be one of the interesting problems in graph theory [Sorlin et al., 2007]. Multivalent matching drops the condition that vertices in the source graph are to be mapped to distinct vertices of the target graph. Thus, in multivalent matching, vertex in the first graph can be matched with an empty set of vertices, one vertex or even multiple vertices in the other graph. Even if the multivalent matching problem has been studied, it is still at the its early stage. Similarly to what we have done for the graph edit distance, mathematical programming and machine learning could be investigated to deal with this problem.

5.2.1.4 Multiple occurrences graph matching

This problem deal with the search of multiple occurrences of a given pattern represented as a input graph within a larger graph. Few works address this problem [Bodic et al., 2012] and usually the methods are iterative and the algorithms are configured so that any vertex mapping in a previous solution is excluded from the search space. At the opposite, the Multi Graph Edit Distance Problem defined in Problem KMGED could be used in a single pass to search for the k most similar matchings and providing a speed up.

Second, we discuss the perspective on the side of learning graph matching.

5.2.2 Learning-based graph matching

We now discuss the possibility to merge combinatorial optimization and machine learning for the purpose of graph matching.

5.2.2.1 Hierarchical feature learning for graph matching methods

We have seen that a big trend is to learn graph matching with graph neural networks. The Siamese architecture for graph matching presented in [Nowak et al., 2017] suffers from a main drawback that no graph matching solver is involved to maintain the constraints consistency. The learning algorithm must learn on its own the combinatorial nature of the problem. At the opposite, in [Zanfir and Sminchisescu, 2018], a feed-forward architecture is proposed and it relies on a graph matching solver. However, the feature extraction is completely domain-dependent and it is dedicated to image processing. Our proposal would be to replace the feature extraction step of [Zanfir and Sminchisescu, 2018] by a Siamese architecture based on graph neural networks. In this way, we would obtain a complete end-to-end graph matching learning scheme. In this direction, we could exploit our graph-matching based neural network. The global architecture is depicted in Figure 5.5. In this strategy, the ML serves as a feature extractor. The extracted features are used to generate an affinity matrix (K) on which operates the graph matching method. Another way of seeing this mechanism is a data generation scheme to facilitate the graph matching. Figure 5.1 illustrates this mechanism.

5.2.2.2 Learning to branch in a branch and bound

In [Zanfir and Sminchisescu, 2018], the graph matching solver does not have any parameter to be learned. The learning stage is located before by extracting features that will facilitate the graph matching stage. In the methods based on structured prediction [Martineau et al., 2018, in press, Raveaux et al., 2017, Cho et al., 2013], ML and CO collaborate together (see Figure 5.2). However, these methods remain shallow. We would like to extend the concept to structured prediction to deep architecture. Especially, we would like to couple structured prediction and a branch bound procedure. In our branch and bound [Abu-Aisheh et al., 2015b], each tree node (p) is either a partial matching plus the remaining nodes/edges to be matched or a feasible matching. We would like to design a predictor as a function f_θ with parameters θ. This predictor would take as an input the tree node p, and outputs both move probabilities and a value, $(m, \hat{opt}) = f_\theta(p)$. The vector of move probabilities m represents the probability of selecting each move (next child node) a, $m_a = Pr(a|p)$. The value $\hat{opt}$ is a scalar evaluation, estimating the probability of the current state p to lead to an optimal solution. At test time, the predictor f_θ will be called by the branch and bound procedure to drive the exploration of the search space to the most promising solution. An illustration is provided in Figure 5.7. In this way, the branch and bound will be equipped with trainable parameters. The predictor f_θ could be a deep architecture such as a GNN.

5.2.3 Learning graph matching for classification

Graph neural networks achieve graph classification by averaging the node embeddings. We have the intuition that this step induces a loose of information. A research direction is to stacked our graph-based perceptron after a graph neural network. Once again, we could exploit our graph-matching based convolution neural network to constantly operate in graph space. The global architecture is depicted in Figure 5.8.

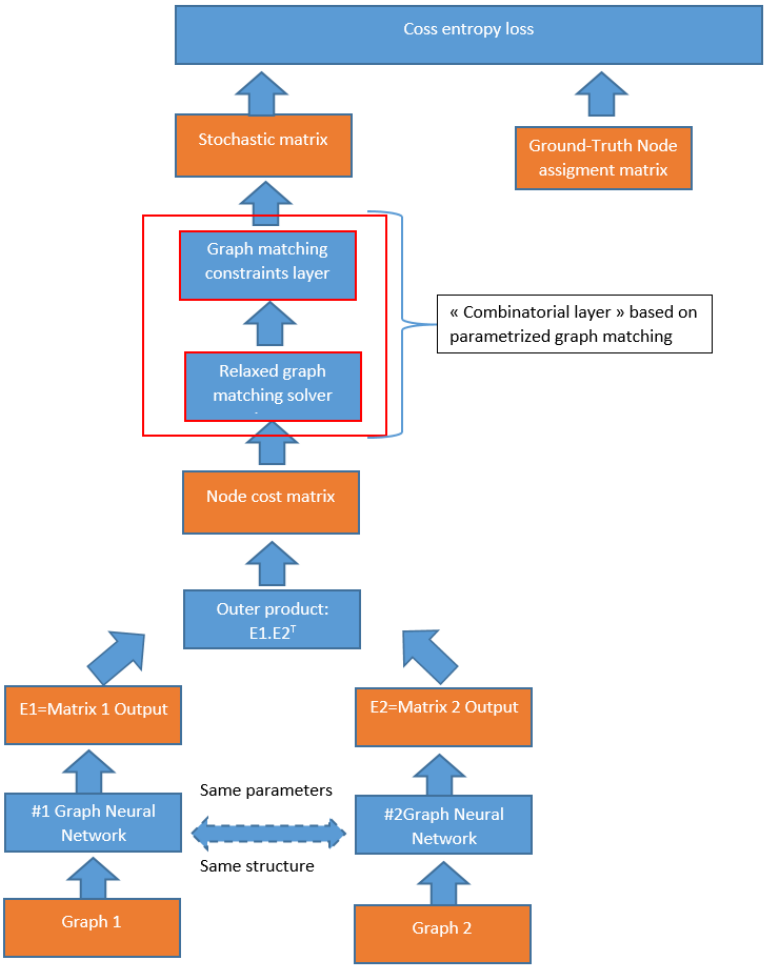


Figure 5.5: A proposal of end-to-end learning graph matching scheme based on a combinatorial layer. Red parts are new components compared to [Nowak et al., 2017]

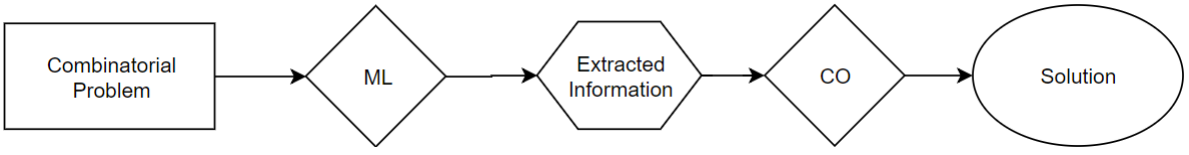


Figure 5.6: The ML method extracts information from the problem. This information (features) is fed to a combinatorial optimization (CO) algorithm.

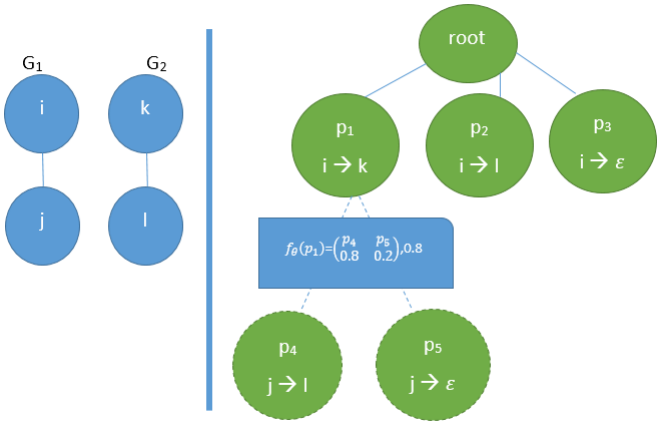


Figure 5.7: A branch and bound algorithm guided by a learned predictor.

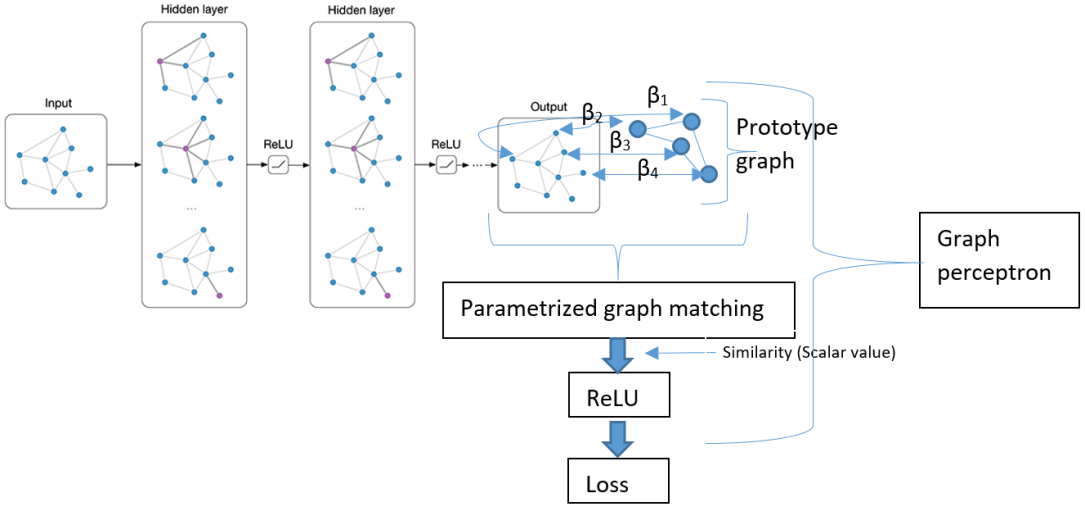


Figure 5.8: A proposal of end-to-end learning graph classification in the graph space. The left part of the image (the GNN) is taken from [Kipf and Welling, 2016]

5.3 Long term perspectives

Now, that we have explained some promising hanging fruits, we can expose some more long term perspectives.

5.3.1 Bringing semantic to computer vision task thanks to graphs

One characteristic that sets humans apart from modern learning-based computer vision algorithms is the ability to acquire knowledge about the world and use that knowledge to reason about the visual world. Humans can learn about the characteristics of objects and the relationships that occur between them to learn a large variety of visual concepts, often with few examples. Our future work will investigate the use of structured prior knowledge in the form of knowledge graphs to improve performance on image classification. Such additional information could help to deal with ambiguity and to bring the context into the classifier. Graph neural networks will be a key component to represent the image and the semantic knowledge about the classification task (classification of documents, scene images or medical images for instance). Graph neural network could be a way of efficiently incorporating knowledge graphs into a vision classification pipeline. A key challenge to require joint reasoning over the visual and knowledge graph domains. The question is even more widely open: how could we built graph neural networks that deal with multiple graphs? Visual question answering is another task where this situation appears. In visual question answering with structured representations of both scene contents and the textual questions are presented by graphs. Structures and features of both graphs are completely different so that they cannot be input to the graph neural network in a straightforward manner. A way to tackle this issue could be to train several neural networks together with a mutual goal. Very promising works in this direction has emerged [Marino et al., 2017, Lee et al., 2018, Teney et al., 2017]. With our insight on graph matching, we could create cost functions for the image graph that depend on the knowledge graph.

5.3.2 On the relation between graph matching and Optimal Transport

Optimal Transport (OT) problems have recently raised interest in several fields, in particular because OT theory can be used for computing distances between probability distributions. These distances have important properties:

- They can be evaluated directly on empirical estimates of the distributions.
- By exploiting the geometry of the underlying metric space, they provide meaningful distances.

Graph matching and OT are related in the sense that both consider interactions (edges) between data.

An optimal transport-like distance [Villani, 2008] can be obtained by comparing the metric spaces directly: It calculates distances between pairs of samples within each domain and measures how these distances compare to those in the other domain. The Gromov-Wasserstein distance is an optimal transport distance and it can be applied to model a graph matching problem : Let G_1, G_2 be two graphs. Let A^1, A^2 be the weighted adjacency matrix of these two graphs. Note that a weight may not be between 0 and 1 but they have to express the similarity between two nodes (the appearances of the interaction). Let $Pm^1 \in \mathbb{R}^{|V_1|}$ be the empirical distribution of nodes of G_1, which counts the appearance of each node in E_1. Pm^2 is built in the same way.

$$d(Pm^1, Pm^2) = \min_{Pr \in \Pi(Pm^1, Pm^2)} \sum_{(i,k) \in V_1 \times V_2} \sum_{(j,l) \in V_1 \times V_2} ||A^1_{i,j} - A^2_{k,l}||.Pr(i,k).Pr(j,l) \qquad (5.3)$$

$Pr \in \mathbb{R}^{|V_1| \times |V_2|}$ is the joint probability of nodes $i \in V_1$ and $k \in V_2$. Pr and Pm^1 are related. Pm^1 is the marginal of Pr such that $Pm^1_i = \sum_{k \in V_2} Pr(i,k)$. $\Pi(Pm^1, Pm^2)$ is the set of all possible joint probabilities Pr that can be obtained from Pm^1 and Pm^2. By choosing the largest $Pr(i,k)$ for each i, the matching that minimizes the Gromov-Wasserstein distance between the two graphs can be obtained. However, the results may not fulfill binary mapping constraints.

Modelling graph matching problems by OT problems is interesting because OT problems have strong solution methods with linear convergence (Sinkhorn-Knopp algorithm [Sinkhorn and Knopp, 1967, Altschuler et al., 2017]). A first step in this direction was made in [Xu et al., 2019] (pre-print of Lawrence Carin).

5.3.3 Graph matching for domain adaptation

Modern data analytics are based on the availability of large volumes of data, sensed by a variety of acquisition devices and at high temporal frequency. But this large amounts of heterogeneous data also make the task of learning semantic concepts more difficult, since the data used for learning a decision function and those used for inference tend not to follow the same distribution. Discrepancies (also known as drift) in data distribution are due to several reasons and are application-dependent. In computer vision, this problem is known as the visual adaptation domain problem, where domain drifts occur when changing lighting conditions, acquisition devices, or by considering the presence or absence of backgrounds. For those reasons, several works have coped with these drift problems by developing learning methods able to transfer knowledge from a source domain to a target domain. Learning in this discrepancy context is denoted as the domain adaptation problem. A variant of domain adaptation is unsupervised domain adaptation, where data labels are only available in the source domain. This problem can be tackled by assuming that the effects of the drifts can be reduced if data undergo a phase of adaptation (typically, a non-linear mapping) where both domains look more alike. The question is then how to transform data so as to make their distributions "closer", and use the label information available in the source domain to learn a classifier in the transformed domain ? If the source domain and the target domain are modeled as graphs then finding matches between samples of the source and target domains can be achieved by graph matching. This process is depicted in Figure 5.9. This idea was investigated in [Das and Lee, 2018] but it could be further developed thanks to graph neural networks. Nodes embeddings and graph matching could be learned in the objective to better classify in the target domain.

5.3.4 The rise of the edge classification

Traditional structural pattern recognition techniques were mainly focused on graph classification and matching. With the rapid emergence of the graph neural networks, new interesting applications come to the surface. In fact graph neural networks provide a common framework to perform classification at node and graph levels. This enables new perspectives such as semantic image segmentation (pixel classification). However, edges are left behind and it is not possible to perform classification at edge level. It could very useful to use our graph matching based neural network for such a purpose because our model outputs node and edge embeddings. A direct applications could be to classify the types of relation between atoms in a molecule graph or to perfom graph factorization.

5.3.5 Benchmarking

This topic is of great importance if we want to take into account the mistakes that have been made in the past within the research on edge detection (some authors proposed many new optimal edge

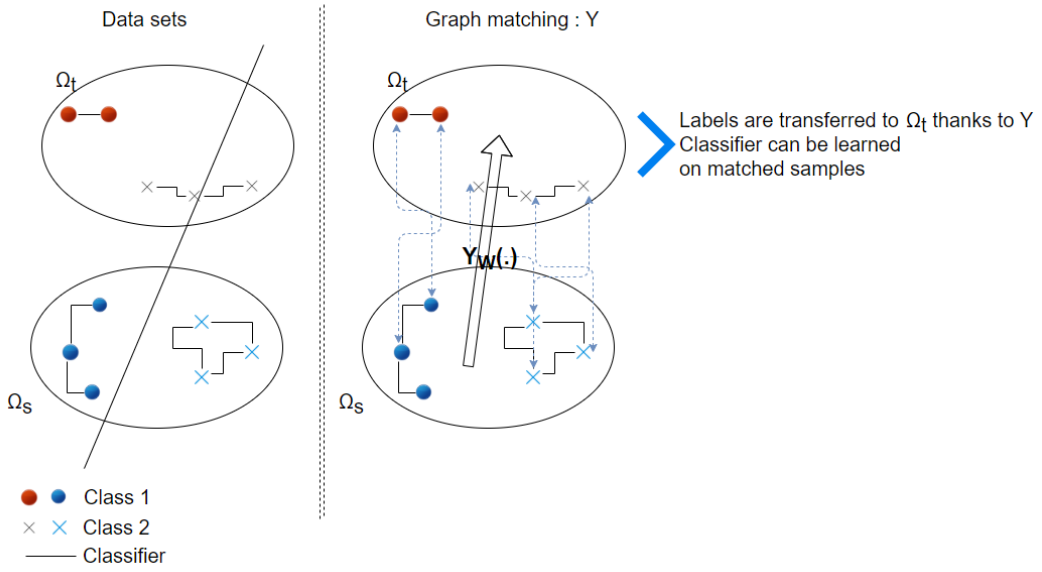


Figure 5.9: Illustration of the domain adaptation. (left) dataset for training, i.e. source domain, and testing, i.e. target domain. Note that a classifier estimated on the training examples clearly does not fit the target data. (middle) a matching Y is estimated and used to transfer the training samples onto the target domain.(right) the transferred labeled samples are used for estimating a classifier in the target domain.

detectors only because we had no tools to make them compete, or just to compare and classify them, on the basis of real data).

5.3.5.1 Learning graph matching

On the side of learning graph matching, none of the papers reported experiments with symbolic attributed graphs. It is true that the focus was given to computer vision methods but is it possible to make all this methods work on graph with discrete attributes? It could be interesting to include in the benchmark graphs with discrete attributes to answer this question. Moreover, running time are rarely reported in the experiments so it is hard to draw conclusion about the scalability of the methods. The amount of data to perform the training is not really discussed. Generally, there is a lack of common benchmarks with precised metrics and various graph data sets.

5.3.5.2 Graph classification

If I would ask myself what is it the best methods to classify graphs, I would get to the point that the answer is not obvious. It would depend on the graphs (density, attributes type, the size) and on the number graphs at hand. There is no clear consensus. Graph neural networks seem very promising but they are not the ultimate weapon. In [Xu et al., 2018], a recent benchmark is proposed to compare graph kernels and graph neural networks. Data sets are made of social networks and molecules. Graph kernels (based random walks) achieve very good results. They are competitive with graph neural networks and sometimes better, especially on molecule graphs. So the debate is not closed. There are rooms to enrich performance evaluation tools and better characterize the methods. This will come up with graphs from different domains (scene images, document images,

medical images, mesh, computer-aided-drawing models, ...) and a common library supporting the different methods would be a plus.

5.3.6 Deep reinforcement learning

Learning from examples (supervised learning) might be undesirable for $\mathcal{NP}$-hard problems because (1) the performance of the model is tied to the quality of the supervised labels, (2) getting high-quality labeled data is expensive and may be infeasible for new problem statements, (3) one cares more about finding a competitive solution more than replicating the results of another algorithm. Reinforcement Learning (RL) is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences [Silver et al., 2016]. Though both supervised and reinforcement learning use the same paradigm but no. Unlike supervised learning where the feedback (the ground-truth label) provided is complete (in term of information) and correct (i.e. $t = [0, 0, 1]$ for a three class problem), reinforcement learning uses rewards and punishment to drive the learning process (i.e. $t = -1$ or 1 to warn the system in case of wrong or good classification, respectively). Standard RL algorithms do not have the ability to estimate values for unseen states. This can be overcome by more advanced algorithms based on deep learning called deep reinforcement learning. Deep reinforcement learning is getting more and more attention thanks to interesting successes. AlphaGo Zero [Silver et al., 2016] is the first computer program to defeat a world champion in the ancient Chinese game of Go. AlphaStar is the first Artificial Intelligence to defeat a top professional player at the Real-Time Strategy game called Starcraft 2. RL requires a lot of (weakly) labelled data, therefore it is most applicable in domains where simulated data. Graph matching problems fall into this category. Many solutions can be generated. Each solution can be evaluated/labelled by its objective function value. The couple RL and deep learning has been applied to solve combinatorial problems such as the traveling salesman problem or knacksack [Bello et al., 2016]. Thanks to the combination of graph neural networks, search tree algorithms and RL, new heuristics for the graph matching problems could appear. One could think about the Siamese architecture [Nowak et al., 2017] and the search tree method [Abu-Aisheh et al., 2015b] trained by RL. The deep reinforcement algorithm would learn how to explore the search tree.

5.3.7 Graph matching for multi-object tracking in videos and document analysis

Tracking multiple objects in videos is an important problem in computer vision which has wide applications in various video analysis scenarios, such as visual surveillance. In particular, we would like to focus on tracking people and cars moving within a video. In this case, category detector can be utilized to facilitate tracking. The major challenge in objects tracking is how to associate noisy detected objects in the current video frame with previously tracked objects. We propose the use of graph-based representations and graph matching to deal with the data association problem.

Many tasks in Pattern Recognition and Document Image Analysis are formulated as graph matching problems. Therefore learning graph-based representations and related techniques is a real interest for the community. Image of documents are likely to be structured because they are images made by humans to humans. Recent work on the problem of subgraph spotting in graph-representation of comic book images (SSGCI) have been published [Le et al., 2018]. We could participate to this applicative problem thanks to our GED methods.

5.3.8 Scalability

5.3.8.1 Graph matching and GPU

Nowadays, when developing graph matching methods to be integrated in a deep neural network, it is important to take into account the explosion of the computing power (GPU and CPU vectorization). GPU compatible methods are more than welcome. Graph matching implementations in a computationally efficient manner thanks to complex matrix calculation is a big challenge.

5.3.8.2 Strategies for matching graphs having large set of nodes

Graph comparison in graph space is a challenging task. Input graphs can be huge. Very powerful tools already exist for matching graphs. However, due to their complexity, these algorithms cannot be used efficiently (in less than a second) for graphs having very large sets of nodes (let us say more than 1000). In such as case, an interesting challenge could be to partition the graph. It could be achieved either by optimization methods based on some graph properties or by learning to partition thanks to paremetrized graph pooling layers for instance. The major advantages would be to create higher level of representations while reducing the computational cost of the graph comparison.

5.3.8.3 Increasing the "intelligence" of a pixel-based graph

When working with image based features, we often end up with a RAG-like graph which is very useful for image analysis but not for image classification, because it takes only into account very low level relations. How can we enhance such a graph without loosing of course its useful properties? A key challenge is then to build methods that can deal directly at pixel levels without data reduction as a preprocessing. Considering Euclidean data like images, graphs can be used to develop non-local approaches and to go beyond the standard 8x8 connectivity. While a pixel is linked to its 8 neighbours in a image, in a graph it is possible to be **non-local** and to extend to neighborhood definition. A pixel can be connected to every pixel in the image and each relation can be enriched by a set of features. As an example the VGG neural network that works well for image classification, its input is a grid of $224 \times 224 = 50176$ pixels. Extending this concept to a kNN graph ($k = 16$ for instance to double the neighborhood compared to a grid) would lead to a graph wit 50176 nodes and 802816 edges. Roughly speaking, this the size of data, we could likely face. A hierarchical method (deep) could then build a higher level of representation based non-local information. Fast graph neural networks will be an important element in the development of such pixel-based approach.

5.4 Synthesis

To conclude, we provide a synthesis of our perspectives under the form of a mind map. Figure 5.10 shows the relations between the key concepts. Short term perspectives are split into two balanced categories combinatorial optimization and machine learning. The long term perspectives are grouped into three parts: application, performance evaluation and fundamental. On the application side, object tracking in videos, domain adaptation and edge classification for graph factorization are targeted. Concerning performance evaluation, the creation of data sets and evaluation metrics as well as scalability tests are considered. Finally, the fundamental aspects concern deep reinforcement learning, optimal transport and semantic computer vision thanks to graphs.

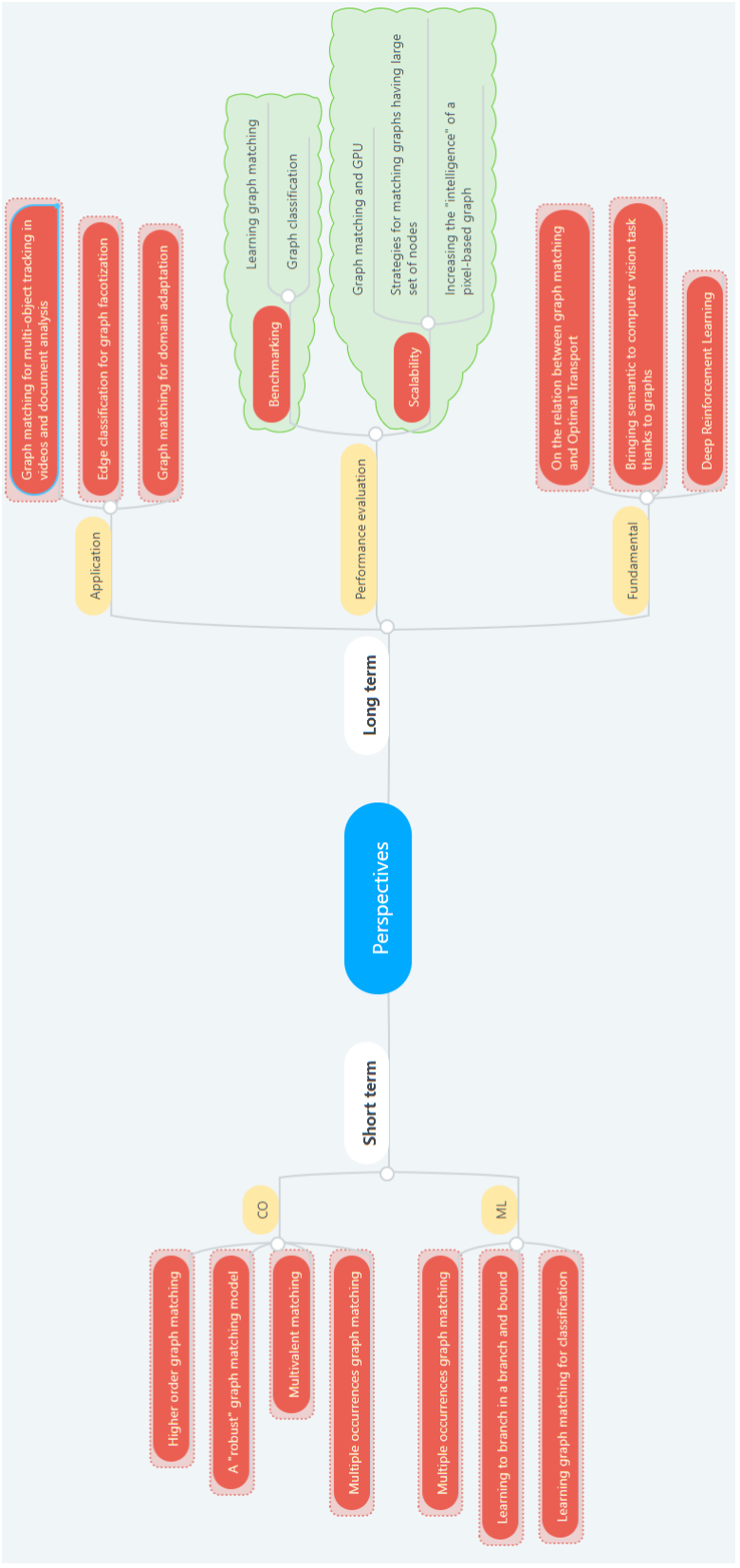


Figure 5.10: A mind map about the perspectives

Bibliography

Cmu house and hotel datasets. URL http://vasc.ri.cmu.edu/idb/html/motion.

Greyc's chemistry dataset. URL https://brunl01.users.greyc.fr/CHEMISTRY/.

18th International Conference on Pattern Recognition (ICPR 2006), 20-24 August 2006, Hong Kong, China, 2006. IEEE Computer Society. ISBN 0-7695-2521-0. URL http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=3995.

Graph flow dataset, 2015. URL http://cvlab-dresden.de/research/image-matching/graphflow/.

K. Fu. A. Sanfeliu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics.*, pages 353–363, 1983.

Zeina Abu-Aisheh. Anytime and distributed approaches for graph matching. *ELCVIA Electronic Letters on Computer Vision and Image Analysis*, 15(2):13–15, 2016. ISSN 1577-5097. URL https://elcvia.cvc.uab.es/article/view/v15-n2-abu-aisheh.

Zeina Abu-Aisheh, Romain Raveaux, and Jean-Yves Ramel. A graph database repository and performance evaluation metrics for graph edit distance. In Cheng-Lin Liu, Bin Luo, Walter G. Kropatsch, and Jian Cheng, editors, *Graph-Based Representations in Pattern Recognition - 10th IAPR-TC-15 International Workshop, GbRPR 2015, Beijing, China, May 13-15, 2015. Proceedings*, volume 9069 of *Lecture Notes in Computer Science*, pages 138–147. Springer, 2015a. ISBN 978-3-319-18223-0. doi:10.1007/978-3-319-18224-7_14. URL https://doi.org/10.1007/978-3-319-18224-7_14.

Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In Maria De Marsico, Mário A. T. Figueiredo, and Ana L. N. Fred, editors, *ICPRAM 2015 - Proceedings of the International Conference on Pattern Recognition Applications and Methods, Volume 1, Lisbon, Portugal, 10-12 January, 2015.*, pages 271–278. SciTePress, 2015b. ISBN 978-989-758-076-5.

Zeina Abu-Aisheh, Romain Raveaux, and Jean-Yves Ramel. Anytime graph matching. *Pattern Recognition Letters*, 84:215–224, 2016. doi:10.1016/j.patrec.2016.10.004. URL https://doi.org/10.1016/j.patrec.2016.10.004.

Zeina Abu-Aisheh, Benoit Gaüzère, Sébastien Bougleux, Jean-Yves Ramel, Luc Brun, Romain Raveaux, Pierre Héroux, and Sébastien Adam. Graph edit distance contest: Results and future challenges. *Pattern Recognition Letters*, 100:96–103, 2017a. doi:10.1016/j.patrec.2017.10.007. URL https://doi.org/10.1016/j.patrec.2017.10.007.

Zeina Abu-Aisheh, Romain Raveaux, and Jean-Yves Ramel. Fast nearest neighbors search in graph space based on a branch-and-bound strategy. In Foggia et al. [2017], pages 197–207. ISBN 978-3-319-58960-2. doi:10.1007/978-3-319-58961-9_18. URL https://doi.org/10.1007/978-3-319-58961-9_18.

Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. A parallel graph edit distance algorithm. *Expert Syst. Appl.*, 94:41–57, 2018. doi:10.1016/j.eswa.2017.10.043. URL https://doi.org/10.1016/j.eswa.2017.10.043.

Zeina Abu-Aisheh, Romain Raveaux, and Jean-Yves Ramel. Efficient k-nearest neighbors search in graph space. *Pattern Recognition Letters*, 2018, in press. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2018.05.001. URL http://www.sciencedirect.com/science/article/pii/S0167865518301673.

Hassan Abu Alhaija, Anita Sellent, Daniel Kondermann, and Carsten Rother. Graphflow – 6d large displacement scene flow via graph matching. In Juergen Gall, Peter Gehler, and Bastian Leibe, editors, *Pattern Recognition*, pages 285–296, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24947-6.

Ammar Ahmed, E. Elhadi, Mohd Aizaini Maarof, and Ahmed Hamza Osman. Malware detection based on hybrid signature behaviour application programming interface call graph. *American Journal of Applied Sciences*, 9(3), 2012.

Alireza Alaei, Donatello Conte, and Romain Raveaux. Document image quality assessment based on improved gradient magnitude similarity deviation. In *13th International Conference on Document Analysis and Recognition, ICDAR 2015, Nancy, France, August 23-26, 2015*, pages 176–180. IEEE Computer Society, 2015. ISBN 978-1-4799-1805-8. doi:10.1109/ICDAR.2015.7333747. URL https://doi.org/10.1109/ICDAR.2015.7333747.

Alireza Alaei, Donatello Conte, Michael Blumenstein, and Romain Raveaux. Document image quality assessment based on texture similarity index. In *12th IAPR Workshop on Document Analysis Systems, DAS 2016, Santorini, Greece, April 11-14, 2016*, pages 132–137. IEEE Computer Society, 2016. ISBN 978-1-5090-1792-8. doi:10.1109/DAS.2016.33. URL https://doi.org/10.1109/DAS.2016.33.

Alireza Alaei, Romain Raveaux, and Donatello Conte. Image quality assessment based on regions of interest. *Signal, Image and Video Processing*, 11(4):673–680, 2017. doi:10.1007/s11760-016-1009-z. URL https://doi.org/10.1007/s11760-016-1009-z.

Alireza Alaei, Donatello Conte, Maxime Martineau, and Romain Raveaux. Blind document image quality prediction based on modification of quality aware clustering method integrating a patch selection strategy. *Expert Syst. Appl.*, 108:183–192, 2018. doi:10.1016/j.eswa.2018.05.007. URL https://doi.org/10.1016/j.eswa.2018.05.007.

H. A. Almohamad and S. O. Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5): 522–525, May 1993. ISSN 0162-8828. doi:10.1109/34.211474.

Jason Altschuler, Jonathan Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1964–1974. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6792-near-linear-time-approximation-algorithms-for-optimal-transport-via-sinkhorn-iteration.pdf.

Kurt M. Anstreicher and Nathan W. Brixius. A new bound for the quadratic assignment problem based on convex quadratic programming. *Mathematical Programming*, 89(3):341–357, Feb 2001. ISSN 1436-4646. doi:10.1007/PL00011402. URL https://doi.org/10.1007/PL00011402.

James Atwood and Don Towsley. Search-convolutional neural networks. *CoRR*, abs/1511.02136, 2015.

László Babai. Graph isomorphism in quasipolynomial time. *CoRR*, abs/1512.03547, 2015.

Eugen Barbu, Romain Raveaux, Hervé Locteau, Sébastien Adam, Pierre Héroux, and Éric Trupin. Graph classification using genetic algorithm and graph probing application to symbol recognition. In *18th International Conference on Pattern Recognition (ICPR 2006), 20-24 August 2006, Hong Kong, China* DBL [2006], pages 296–299. ISBN 0-7695-2521-0. doi:10.1109/ICPR.2006.612. URL https://doi.org/10.1109/ICPR.2006.612.

Gustavo E. A. P. A. Batista and Diego F. Silva. How k-nearest neighbor parameters affect its performance. In *Argentine Symposium on Artificial Intelligence*, pages 95–106, 2009.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016.

Endika Bengoetxea, Pedro Larrañaga, Isabelle Bloch, and Aymeric Perchant. Estimation of distribution algorithms: A new evolutionary computation approach for graph matching problems. In Mário Figueiredo, Josiane Zerubia, and Anil K. Jain, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 454–469, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44745-0.

Endika Bengoetxea, Pedro Larrañaga, Isabelle Bloch, Aymeric Perchant, and Claudia Boeres. Inexact graph matching by means of estimation of distribution algorithms. *Pattern Recognition*, 35(12):2867 – 2880, 2002. ISSN 0031-3203. doi:https://doi.org/10.1016/S0031-3203(01)00232-1. URL http://www.sciencedirect.com/science/article/pii/S0031320301002321. Pattern Recognition in Information Systems.

Nitin Bhatia and Vandana. Survey of nearest neighbor techniques. *CoRR*, abs/1007.0085, 2010. URL http://arxiv.org/abs/1007.0085.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008 (10):P10008, 2008. URL http://stacks.iop.org/1742-5468/2008/i=10/a=P10008.

Pierre Le Bodic, Pierre Héroux, Sébastien Adam, and Yves 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. doi:10.1016/j.patcog.2012.05.022. URL https://doi.org/10.1016/j.patcog.2012.05.022.

Ehsan Zare Borzeshi, Massimo Piccardi, Kaspar Riesen, and Horst Bunke. Discriminative prototype selection methods for graph embedding. *Pattern Recognition*, 46(6):1648 – 1657, 2013. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2012.11.020. URL http://www.sciencedirect.com/science/article/pii/S0031320312005043.

Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. *CoRR*, abs/1605.06437, 2016.

Sébastien Bougleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzère, and Mario Vento. Graph edit distance as a quadratic assignment problem. *Pattern Recognition Letters*, 87: 38 – 46, 2017a. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2016.10.001. URL http://www.sciencedirect.com/science/article/pii/S0167865516302665. Advances in Graph-based Pattern Recognition.

Sébastien Bougleux, Benoit Gaüzère, and Luc Brun. A hungarian algorithm for error-correcting graph matching. In Pasquale Foggia, Cheng-Lin Liu, and Mario Vento, editors, *Graph-Based Representations in Pattern Recognition*, pages 118–127, Cham, 2017b. Springer International Publishing. ISBN 978-3-319-58961-9.

Martial Bourquin, Andy King, and Edward Robbins. Binslayer: Accurate comparison of binary executables. In *Proceedings of the 2Nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop*, PPREW '13, pages 4:1–4:10, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1857-0. doi:10.1145/2430553.2430557. URL http://doi.acm.org/10.1145/2430553.2430557.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *CoRR*, abs/1611.08097, 2016.

Luc Brun, Benoit Gaüzère, and Sébastien Fourey. Relationships between Graph Edit Distance and Maximal Common Unlabeled Subgraph. Technical report, July 2012. URL https://hal.archives-ouvertes.fr/hal-00714879.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203, 2013.

Horst Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997. doi:10.1016/S0167-8655(97)00060-3. URL https://doi.org/10.1016/S0167-8655(97)00060-3.

Horst Bunke. Error correcting graph matching: On the influence of the underlying cost function. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(9):917–922, 1999. doi:10.1109/34.790431. URL https://doi.org/10.1109/34.790431.

Horst Bunke and Kaspar Riesen. Graph classification on dissimilarity space embedding. In *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshop, SSPR & SPR 2008, Orlando, USA, December 4-6, 2008. Proceedings*, page 2, 2008. doi:10.1007/978-3-540-89689-0_2. URL http://dx.doi.org/10.1007/978-3-540-89689-0_2.

Horst Bunke and Kaspar Riesen. Towards the unification of structural and statistical pattern recognition. *Pattern Recognition Letters*, 33(7):811 – 825, 2012. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2011.04.017. URL http://www.sciencedirect.com/science/article/pii/S0167865511001309. Special Issue on Awards from ICPR 2010.

Rainer E. Burkard, Eranda Çela, Panos M. Pardalos, and Leonidas S. Pitsoulis. *Handbook of Combinatorial Optimization: Volume1-3*, chapter The Quadratic Assignment Problem, pages 1713–1809. Springer US, Boston, MA, 1999. ISBN 978-1-4613-0303-9. URL https://doi.org/10.1007/978-1-4613-0303-9_27.

Tibério S. Caetano, Julian John McAuley, Li Cheng, Quoc V. Le, and Alexander J. Smola. Learning graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(6):1048–1058, 2009.

HongYun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques and applications. *CoRR*, abs/1709.07604, 2017. URL http://arxiv.org/abs/1709.07604.

Ramzi Chaieb, Karim Kalti, Muhammad Muzzamil Luqman, Mickaël Coustaty, Jean-Marc Ogier, and Najoua Essoukri Ben Amara. Fuzzy generalized median graphs computation: Application to content-based document retrieval. *Pattern Recognition*, 72:266 – 284, 2017. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2017.07.030. URL http://www.sciencedirect.com/science/article/pii/S0031320317303011.

M. Cho, J. Sun, O. Duchenne, and J. Ponce. Finding matches in a haystack: A max-pooling strategy for graph matching in the presence of outliers. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2091–2098, June 2014. doi:10.1109/CVPR.2014.268.

Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 492–505, Berlin, Heidelberg, 2010a. Springer Berlin Heidelberg. ISBN 978-3-642-15555-0.

Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 492–505, Berlin, Heidelberg, 2010b. Springer Berlin Heidelberg. ISBN 978-3-642-15555-0.

Minsu Cho, Karteek Alahari, and Jean Ponce. Learning graphs to match. In *IEEE International Conference on Computer Vision, ICCV 2013*, pages 25–32, 2013.

Yeonjoo Choi and Gyeonghwan Kim. Graph-based fingerprint classification using orientation field in core area. *IEICE Electronics Express*, 7(17):1303–1309, 2010. doi:10.1587/elex.7.1303.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi:10.3115/1118693.1118694. URL https://doi.org/10.3115/1118693.1118694.

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*, 18(03):265–298, 2004. doi:10.1142/S0218001404003228. URL https://doi.org/10.1142/S0218001404003228.

Xavier Cortés and Francesc Serratosa. Learning graph-matching edit-costs based on the optimality of the oracle's node correspondences. *Pattern Recogn. Lett.*, 56:22–29, 2015.

Xavier Cortés and Francesc Serratosa. Learning graph matching substitution weights based on the ground truth node correspondence. *IJPRAI*, 30(2), 2016.

Timothee Cour, Praveen Srinivasan, and Jianbo Shi. Balanced graph matching. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 313–320. MIT Press, 2007. URL http://papers.nips.cc/paper/2960-balanced-graph-matching.pdf.

Mickaël Coustaty, Romain Raveaux, and Jean-Marc Ogier. Historical document analysis: A review of french projects and open issues. In *Proceedings of the 19th European Signal Processing Conference, EUSIPCO 2011, Barcelona, Spain, August 29 - Sept. 2, 2011*, pages 1445–1449. IEEE, 2011. URL http://ieeexplore.ieee.org/document/7074005/.

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967. ISSN 0018-9448. doi:10.1109/TIT.1967.1053964.

Andrew D.J. Cross, Richard C. Wilson, and Edwin R. Hancock. Inexact graph matching using genetic search. *Pattern Recognition*, 30(6):953 – 970, 1997. ISSN 0031-3203. doi:https://doi.org/10.1016/S0031-3203(96)00123-9. URL http://www.sciencedirect.com/science/article/pii/S0031320396001239.

Mostafa Darwiche, Romain Raveaux, Donatello Conte, and Vincent T'Kindt. A local branching heuristic for the graph edit distance problem. In Marcelo Mendoza and Sergio A. Velastin, editors, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*

- *22nd Iberoamerican Congress, CIARP 2017, Valparaíso, Chile, November 7-10, 2017, Proceedings*, volume 10657 of *Lecture Notes in Computer Science*, pages 194–202. Springer, 2017. ISBN 978-3-319-75192-4. doi:10.1007/978-3-319-75193-1_24. URL https://doi.org/10.1007/978-3-319-75193-1_24.

Mostafa Darwiche, Donatello Conte, Romain Raveaux, and Vincent T'Kindt. A local branching heuristic for solving a graph edit distance problem. *Computers and Operations Research*, 2018, in pressa. ISSN 0305-0548. doi:https://doi.org/10.1016/j.cor.2018.02.002. URL http://www.sciencedirect.com/science/article/pii/S0305054818300339.

Mostafa Darwiche, Donatello Conte, Romain Raveaux, and Vincent T'Kindt. Graph edit distance: Accuracy of local branching from an application point of view. *Pattern Recognition Letters*, 2018, in pressb. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2018.03.033. URL http://www.sciencedirect.com/science/article/pii/S0167865518301119.

Debasmit Das and C.S. George Lee. Sample-to-sample correspondence for unsupervised domain adaptation. *Engineering Applications of Artificial Intelligence*, 73:80 – 91, 2018. ISSN 0952-1976. doi:https://doi.org/10.1016/j.engappai.2018.05.001. URL http://www.sciencedirect.com/science/article/pii/S0952197618301088.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016.

Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(11):1944–1957, November 2007. ISSN 0162-8828. doi:10.1109/TPAMI.2007.1115. URL http://dx.doi.org/10.1109/TPAMI.2007.1115.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2224–2232. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5954-convolutional-networks-on-graphs-for-learning-molecular-fingerprints.pdf.

M. Ferrer, E. Valveny, and F. Serratosa. Median graphs: A genetic approach based on new theoretical properties. *Pattern Recognition*, 42(9):2003 – 2012, 2009. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2009.01.034. URL http://www.sciencedirect.com/science/article/pii/S003132030900065X.

M. Ferrer, E. Valveny, F. Serratosa, K. Riesen, and H. Bunke. Generalized median graph computation by means of graph embedding in vector spaces. *Pattern Recognition*, 43(4):1642 – 1655, 2010. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2009.10.013. URL http://www.sciencedirect.com/science/article/pii/S0031320309003860.

M. Ferrer, D. Karatzas, E. Valveny, I. Bardaji, and H. Bunke. A generic framework for median graph computation based on a recursive embedding approach. *Computer Vision and Image Understanding*, 115(7):919 – 928, 2011. ISSN 1077-3142. doi:https://doi.org/10.1016/j.cviu.2010.12.010. URL http://www.sciencedirect.com/science/article/pii/S1077314211000786. Special issue on Graph-Based Representations in Computer Vision.

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

Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1):23–47, Sep 2003. ISSN 1436-4646. doi:10.1007/s10107-003-0395-5. URL https://doi.org/10.1007/s10107-003-0395-5.

M. A. Fischler and R. A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, C-22(1):67–92, Jan 1973. ISSN 0018-9340. doi:10.1109/T-C.1973.223602.

Pasquale Foggia, Cheng-Lin Liu, and Mario Vento, editors. *Graph-Based Representations in Pattern Recognition - 11th IAPR-TC-15 International Workshop, GbRPR 2017, Anacapri, Italy, May 16-18, 2017, Proceedings*, volume 10310 of *Lecture Notes in Computer Science*, 2017. ISBN 978-3-319-58960-2. doi:10.1007/978-3-319-58961-9. URL https://doi.org/10.1007/978-3-319-58961-9.

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110. doi:10.1002/nav.3800030109. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030109.

Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and Applications*, 13(1):113–129, Feb 2010. ISSN 1433-755X. doi:10.1007/s10044-008-0141-y. URL https://doi.org/10.1007/s10044-008-0141-y.

Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. ISBN 0716710455.

Thomas Gärtner. A survey of kernels for structured data. *SIGKDD Explor. Newsl.*, 5(1):49–58, 2003. ISSN 1931-0145. doi:http://doi.acm.org/10.1145/959242.959248. URL http://portal.acm.org/citation.cfm?doid=959242.959248.

Benoit Gaüzère, Luc Brun, and Didier Villemin. Two new graphs kernels in chemoinformatics. *Pattern Recognition Letters*, 33(15):2038–2047, 2012. doi:10.1016/j.patrec.2012.03.020. URL https://doi.org/10.1016/j.patrec.2012.03.020.

S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, April 1996. ISSN 0162-8828. doi:10.1109/34.491619.

Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bull. Amer. Math. Soc.*, 64(5):275–278, 09 1958. URL https://projecteuclid.org:443/euclid.bams/1183522679.

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *CoRR*, abs/1705.02801, 2017. URL http://arxiv.org/abs/1705.02801.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

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

Peter Hahn and Thomas Grant. Lower bounds for the quadratic assignment problem based upon a dual formulation. *Operations Research*, 46(6):912–922, 1998. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/222943.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

Edwin R. Hancock and Richard C. Wilson. Pattern analysis with graphs: Parallel work at bern and york. *Pattern Recognition Letters*, 33(7):833 – 841, 2012. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2011.08.012. URL http://www.sciencedirect.com/science/article/pii/S0167865511002637. Special Issue on Awards from ICPR 2010.

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *CoRR*, abs/1506.05163, 2015.

Rashid Ibragimov, Maximilian Malek, Jiong Guo, and Jan Baumbach. GEDEVO: An Evolutionary Graph Edit Distance Algorithm for Biological Network Alignment. In Tim Beißbarth, Martin Kollmar, Andreas Leha, Burkhard Morgenstern, Anne-Kathrin Schultz, Stephan Waack, and Edgar Wingender, editors, *German Conference on Bioinformatics 2013*, volume 34 of *OpenAccess Series in Informatics (OASIcs)*, pages 68–79, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-59-0. doi:10.4230/OASIcs.GCB.2013.68. URL http://drops.dagstuhl.de/opus/volltexte/2013/4229.

Xiaoyi Jiang, Andreas Münger, and Horst Bunke. On median graphs: Properties, algorithms, and applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(10):1144–1151, 2001.

Hyung Jin Chang and Yiannis Demiris. Unsupervised learning of complex articulated kinematic structures combining motion and skeleton information. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, Oct 2009. ISSN 1573-0565. doi:10.1007/s10994-009-5108-8. URL https://doi.org/10.1007/s10994-009-5108-8.

D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1200–1214, Aug 2006. ISSN 0162-8828. doi:10.1109/TPAMI.2006.152.

J. H. Kappes, B. Savchynskyy, and C. Schnörr. A bundle approach to efficient map-inference by lagrangian relaxation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1688–1695, June 2012. doi:10.1109/CVPR.2012.6247863.

Stefan E. Karisch and Franz Rendl. Lower bounds for the quadratic assignment problem via triangle decompositions. *Mathematical Programming*, 71(2):137–151, Dec 1995. ISSN 1436-4646. doi:10.1007/BF01585995. URL https://doi.org/10.1007/BF01585995.

Leonard Kaufman and Peter J. Rousseeuw. *Clustering by means of medoids*. North Holland / Elsevier, 1987.

Maurice G. Kendall. *Rank correlation methods*. Griffin, London, 1948. URL http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+18489199X&sourceid=fbw_bibsonomy.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

Philip A. Knight. The sinkhorn-knopp algorithm: Convergence and applications. *SIAM J. Matrix Analysis Applications*, 30:261–275, 2007.

V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, Oct 2006. ISSN 0162-8828. doi:10.1109/TPAMI.2006.200.

Tjalling Koopmans and Martin J. Beckmann. Assignment problems and the location of economic activities. Cowles Foundation Discussion Papers 4, Cowles Foundation for Research in Economics, Yale University, 1955. URL https://EconPapers.repec.org/RePEc:cwl:cwldpp:4.

Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Publishing Company, Incorporated, 4th edition, 2007. ISBN 3540718435, 9783540718437.

H. W. Kuhn and Bryn Yaw. The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.

Neeraj Kumar, Li Zhang, and Shree Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision – ECCV 2008*, pages 364–378, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-88688-4.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL http://dl.acm.org/citation.cfm?id=645530.655813.

Eugene L. Lawler. The quadratic assignment problem. *Management Science*, 9(4):586–599, 1963. URL https://EconPapers.repec.org/RePEc:inm:ormnsc:v:9:y:1963:i:4:p:586-599.

Thanh Nam Le, Muhammad Muzzamil Luqman, Anjan Dutta, Pierre Héroux, Christophe Rigaud, Clément Guérin, Pasquale Foggia, Jean-Christophe Burie, Jean-Marc Ogier, Josep Lladós, and Sébastien Adam. Subgraph spotting in graph representations of comic book images. *Pattern Recognition Letters*, 112:118 – 124, 2018. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2018.06.017. URL http://www.sciencedirect.com/science/article/pii/S0167865518302629.

Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

Chung-Wei Lee, Wei Fang, Chih-Kuan Yeh, and Yu-Chiang Frank Wang. Multi-label zero-shot learning with structured knowledge graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1482–1489 Vol. 2, Oct 2005. doi:10.1109/ICCV.2005.20.

Marius Leordeanu and Martial Hebert. Unsupervised learning for graph matching. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 864–871. IEEE Computer Society, 2009. ISBN 978-1-4244-3992-8. doi:10.1109/CVPRW.2009.5206533. URL https://doi.org/10.1109/CVPRW.2009.5206533.

Marius Leordeanu, Martial Hebert, and Rahul Sukthankar. An integer projected fixed point method for graph matching and map inference. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1114–1122. Curran Associates, Inc., 2009. URL http://papers.nips.cc/paper/3756-an-integer-projected-fixed-point-method-for-graph-matching-and-map-inference.pdf.

Marius Leordeanu, Rahul Sukthankar, and Martial Hebert. Unsupervised learning for graph matching. *International Journal of Computer Vision*, 96(1):28–45, Jan 2012. ISSN 1573-1405. doi: 10.1007/s11263-011-0442-2. URL https://doi.org/10.1007/s11263-011-0442-2.

Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. Exact graph edit distance computation using a binary linear program. In Antonio Robles-Kelly, Marco Loog, Battista Biggio, Francisco Escolano, and Richard C. Wilson, editors, *Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop, S+SSPR 2016, Mérida, Mexico, November 29 - December 2, 2016, Proceedings*, volume 10029 of *Lecture Notes in Computer Science*, pages 485–495, 2016. ISBN 978-3-319-49054-0. doi: 10.1007/978-3-319-49055-7_43. URL https://doi.org/10.1007/978-3-319-49055-7_43.

Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. New binary linear programming formulation to compute the graph edit distance. *Pattern Recognition*, 72:254–265, 2017. doi:10.1016/j.patcog.2017.07.029. URL https://doi.org/10.1016/j.patcog.2017.07.029.

Samuel Leturcq and Romain Raveaux. Les graphes pour étudier les dynamiques spatiales à partir des séries fiscales médiévales et modernes. Etat des lieux de l'expérience Modelespace. *Bulletin du Centre d'études médiévales d'Auxerre. Hors-série*, (9), April 2016. doi:10.4000/cem.13805. URL https://hal.archives-ouvertes.fr/hal-01311067.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. *arXiv e-prints*, art. arXiv:1511.05493, November 2015.

C. Karen Liu, Aaron Hertzmann, and Zoran Popović. Learning physics-based motion style with nonlinear inverse optimization. *ACM Trans. Graph.*, 24(3):1071–1081, July 2005. ISSN 0730-0301. doi:10.1145/1073204.1073314. URL http://doi.acm.org/10.1145/1073204.1073314.

Z. Liu and H. Qiao. Gnccp—graduated nonconvexityand concavity procedure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(6):1258–1267, June 2014. ISSN 0162-8828. doi:10.1109/TPAMI.2013.223.

Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, and Le Song. Geniepath: Graph neural networks with adaptive receptive paths. *CoRR*, abs/1802.00910, 2018.

Hervé Locteau, Romain Raveaux, Sébastien Adam, Yves Lecourtier, Pierre Héroux, and Éric Trupin. Polygonal approximation of digital curves using a multi-objective genetic algorithm. In Wenyin Liu and Josep Lladós, editors, *Graphics Recognition. Ten Years Review and Future Perspectives, 6th Internation Workshop, GREC 2005, Hong Kong, China, August 25-26, 2005, Revised Selected Papers*, volume 3926 of *Lecture Notes in Computer Science*, pages 300–311. Springer, 2005. ISBN 978-3-540-34711-8. doi:10.1007/11767978_27. URL https://doi.org/10.1007/11767978_27.

Hervé Locteau, Romain Raveaux, Sébastien Adam, Yves Lecourtier, Pierre Héroux, and Éric Trupin. Approximation of digital curves using a multi-objective genetic algorithm. In *18th International Conference on Pattern Recognition (ICPR 2006), 20-24 August 2006, Hong Kong, China* DBL [2006], pages 716–719. ISBN 0-7695-2521-0. doi:10.1109/ICPR.2006.276. URL https://doi.org/10.1109/ICPR.2006.276.

Muhammad Muzzamil Luqman, Jean-Yves Ramel, Josep Lladós, and Thierry Brouard. Fuzzy multilevel graph embedding. *Pattern Recognition*, 46(2):551–565, 2013. doi:10.1016/j.patcog.2012.07.029. URL http://dx.doi.org/10.1016/j.patcog.2012.07.029.

V. Lyzinski, D. E. Fishkind, M. Fiori, J. T. Vogelstein, C. E. Priebe, and G. Sapiro. Graph matching: Relax at your own risk. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(1):60–73, Jan 2016. ISSN 0162-8828. doi:10.1109/TPAMI.2015.2424894.

Yury A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *CoRR*, abs/1603.09320, 2016. URL http://arxiv.org/abs/1603.09320.

Gaétan Marceau-Caron and Yann Ollivier. Practical riemannian neural networks. *CoRR*, abs/1602.08007, 2016. URL http://arxiv.org/abs/1602.08007.

Kenneth Marino, Ruslan Salakhutdinov, and Abhinav Gupta. The more you know: Using knowledge graphs for image classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Maxime Martineau, Donatello Conte, Romain Raveaux, Ingrid Arnault, Damien Munier, and Gilles Venturini. A survey on image-based insect classification. *Pattern Recognition*, 65:273–284, 2017. doi:10.1016/j.patcog.2016.12.020. URL https://doi.org/10.1016/j.patcog.2016.12.020.

Maxime Martineau, Romain Raveaux, Donatello Conte, and Gilles Venturini. Learning error-correcting graph matching with a multiclass neural network. *Pattern Recognition Letters*, 2018, in press. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2018.03.031. URL http://www.sciencedirect.com/science/article/pii/S0167865518301107.

D. Mateus, R. Horaud, D. Knossow, F. Cuzzolin, and E. Boyer. Articulated shape matching using laplacian eigenfunctions and unsupervised point registration. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008. doi:10.1109/CVPR.2008.4587538.

B.T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(5):493–504, 1998.

Nenad Mladenovic, Jack Brimberg, Pierre Hansen, and José A. Moreno-Pérez. The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, 179(3):927 – 939, 2007. ISSN 0377-2217. doi:https://doi.org/10.1016/j.ejor.2005.05.034. URL http://www.sciencedirect.com/science/article/pii/S0377221706000750.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *CoRR*, abs/1611.08402, 2016.

Carlos Francisco Moreno-García, Xavier Cortés, and Francesc Serratosa. A graph repository for learning error-tolerant graph matching. In *SSPR*, pages 519–529, 2016.

Leonardo M. Musmanno and Celso C. Ribeiro. Heuristics for the generalized median graph problem. *European Journal of Operational Research*, 254(2):371 – 384, 2016. ISSN 0377-2217. doi:https://doi.org/10.1016/j.ejor.2016.03.048. URL http://www.sciencedirect.com/science/article/pii/S0377221716301941.

R. Myers, R. C. Wison, and E. R. Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):628–635, June 2000. ISSN 0162-8828. doi:10.1109/34.862201.

J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965. doi:10.1093/comjnl/7.4.308. URL http://dx.doi.org/10.1093/comjnl/7.4.308.

M. Neuhaus and H. Bunke. Bridging the gap between graph edit distance and kernel machines. *Machine Perception and Artificial Intelligence.*, 68:17–61, 2007.

Michel Neuhaus and Horst Bunke. Self-organizing maps for learning the edit costs in graph matching. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 35(3):503–514, 2005.

Michel Neuhaus and Horst Bunke. Automatic learning of cost functions for graph edit distance. *Inf. Sci.*, 177(1):239–247, 2007.

Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In Dit-Yan Yeung, James T. Kwok, Ana L. N. Fred, Fabio Roli, and Dick de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshops, SSPR 2006 and SPR 2006, Hong Kong, China, August 17-19, 2006, Proceedings*, volume 4109 of *Lecture Notes in Computer Science*, pages 163–172. Springer, 2006. ISBN 3-540-37236-9. doi:10.1007/11815921_17. URL https://doi.org/10.1007/11815921_17.

Alex Nowak, Soledad Villar, Afonso S. Bandeira, and Joan Bruna. Revised Note on Learning Algorithms for Quadratic Assignment with Graph Neural Networks. *arXiv e-prints*, art. arXiv:1706.07450, June 2017.

Alex Nowak, Soledad Villar, Afonso S. Bandeira, and Joan Bruna. A note on learning algorithms for quadratic assignment with graph neural networks. *CoRR*, abs/1706.07450, 2017. URL http://arxiv.org/abs/1706.07450.

E. Ozdemir and C. Gunduz-Demir. A hybrid classification model for digital pathology using structural and statistical pattern recognition. *IEEE Transactions on Medical Imaging*, 32(2): 474–483, Feb 2013. ISSN 0278-0062. doi:10.1109/TMI.2012.2230186.

Jiahao Pang and Gene Cheung. Graph laplacian regularization for inverse imaging: Analysis in the continuous domain. *CoRR*, abs/1604.07948, 2016.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014.

Florian A. Potra and Stephen J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1):281 – 302, 2000. ISSN 0377-0427. doi:https://doi.org/10.1016/S0377-0427(00)00433-7. URL http://www.sciencedirect.com/science/article/pii/S0377042700004337. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011. ISBN 1107015359, 9781107015357.

Romain Raveaux. *Graph Mining and Graph Classification : application to cadastral map analysis*. Theses, Université de La Rochelle, November 2010. URL https://tel.archives-ouvertes.fr/tel-00567218.

Romain Raveaux and Guillaume Hillairet. Model driven image segmentation using a genetic algorithm for structured data. In Manuel Graña Romay, Emilio Corchado, and M. Teresa García-Sebastián, editors, *Hybrid Artificial Intelligence Systems, 5th International Conference, HAIS 2010, San Sebastián, Spain, June 23-25, 2010. Proceedings, Part I*, volume 6076 of *Lecture Notes in Computer Science*, pages 311–318. Springer, 2010. ISBN 978-3-642-13768-6. doi:10.1007/978-3-642-13769-3_38. URL https://doi.org/10.1007/978-3-642-13769-3_38.

Romain Raveaux, Eugen Barbu, Hervé Locteau, Sébastien Adam, Pierre Héroux, and Éric Trupin. A graph classification approach using a multi-objective genetic algorithm application to symbol recognition. In Francisco Escolano and Mario Vento, editors, *Graph-Based Representations in Pattern Recognition, 6th IAPR-TC-15 International Workshop, GbRPR 2007, Alicante, Spain, June 11-13, 2007, Proceedings*, volume 4538 of *Lecture Notes in Computer Science*, pages 361–370. Springer, 2007a. ISBN 978-3-540-72902-0. doi:10.1007/978-3-540-72903-7_33. URL https://doi.org/10.1007/978-3-540-72903-7_33.

Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A segmentation scheme based on a multi-graph representation: Application to colour cadastral maps. In Wenyin Liu, Josep Lladós, and Jean-Marc Ogier, editors, *Graphics Recognition. Recent Advances and New Opportunities, 7th International Workshop, GREC 2007, Curitiba, Brazil, September 20-21, 2007. Selected Papers*, volume 5046 of *Lecture Notes in Computer Science*, pages 202–212. Springer, 2007b. ISBN 978-3-540-88184-1. doi:10.1007/978-3-540-88188-9_20. URL https://doi.org/10.1007/978-3-540-88188-9_20.

Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A colour document interpretation: Application to ancient cadastral maps. In *9th International Conference on Document Analysis and Recognition (ICDAR 2007), 23-26 September, Curitiba, Paraná, Brazil*, pages 1128–1132. IEEE Computer Society, 2007c. ISBN 978-0-7695-2822-9. doi:10.1109/ICDAR.2007.5. URL http://doi.ieeecomputersociety.org/10.1109/ICDAR.2007.5.

Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A contribution to ancient cadastral maps interpretation through colour analysis. In Ana L. N. Fred and Anil K. Jain, editors, *Pattern Recognition in Information Systems, Proceedings of the 7th International Workshop on Pattern Recognition in Information Systems, PRIS 2007, In conjunction with ICEIS 2007, Funchal, Madeira, Portugal, June 2007*, pages 89–98. INSTICC PRESS, 2007d. ISBN 978-972-8865-93-1.

Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. Object extraction from colour cadastral maps. In Koichi Kise and Hiroshi Sako, editors, *The Eighth IAPR International Workshop on Document Analysis Systems, DAS 2008, September 16-19, 2008, Nara, Japan*, pages 506–514. IEEE Computer Society, 2008a. ISBN 978-0-7695-3337-7. doi:10.1109/DAS.2008.9. URL https://doi.org/10.1109/DAS.2008.9.

Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A colour text/graphics separation based on a graph representation. In *19th International Conference on Pattern Recognition (ICPR 2008), December 8-11, 2008, Tampa, Florida, USA*, pages 1–4. IEEE Computer Society, 2008b. ISBN 978-1-4244-2175-6. doi:10.1109/ICPR.2008.4761725. URL https://doi.org/10.1109/ICPR.2008.4761725.

Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A colour space selection scheme dedicated to information retrieval tasks. In Alfons Juan-Císcar and Gemma Sánchez-Albaladejo, editors, *Pattern Recognition in Information Systems, Proceedings of the 8th International Workshop on Pattern Recognition in Information Systems, PRIS 2008, In conjunction with ICEIS 2008, Barcelona, Spain, June 2008*, pages 123–134. INSTICC PRESS, 2008c. ISBN 978-989-8111-42-5.

Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A graph matching method and a graph matching distance based on subgraph assignments. *Pattern Recognition Letters*, 31(5): 394–406, 2010. doi:10.1016/j.patrec.2009.10.011. URL https://doi.org/10.1016/j.patrec.2009.10.011.

Romain Raveaux, Sébastien Adam, Pierre Héroux, and Éric Trupin. Learning graph prototypes for shape recognition. *Computer Vision and Image Understanding*, 115(7):905–918, 2011. doi:10.1016/j.cviu.2010.12.015. URL https://doi.org/10.1016/j.cviu.2010.12.015.

Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A local evaluation of vectorized documents by means of polygon assignments and matching. *IJDAR*, 15(1):21–43, 2012. doi:10.1007/s10032-010-0143-3. URL https://doi.org/10.1007/s10032-010-0143-3.

Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. Structured representations in a content based image retrieval context. *J. Visual Communication and Image Representation*, 24(8):1252–1268, 2013a. doi:10.1016/j.jvcir.2013.08.010. URL https://doi.org/10.1016/j.jvcir.2013.08.010.

Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. La vectorisation automatisée des plans Vasserot, dans Paris de parcelles en pixels. In *Analyse géomatique de l'espace parisien médiéval et moderne*, pages 54–65. Presse Universitaire de Vincennes, April 2013b. URL https://hal.archives-ouvertes.fr/hal-00936555.

Romain Raveaux, Maxime Martineau, Donatello Conte, and Gilles Venturini. Learning graph matching with a graph-based perceptron in a classification context. In Foggia et al. [2017], pages 49–58. ISBN 978-3-319-58960-2. doi:10.1007/978-3-319-58961-9_5. URL https://doi.org/10.1007/978-3-319-58961-9_5.

John W. Raymond and Peter Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16(7):521–533, Jul 2002. ISSN 1573-4951. doi:10.1023/A:1021271615909. URL https://doi.org/10.1023/A:1021271615909.

Pau Riba, Andreas Fischer, Josep Lladós, and Alicia Fornés. Learning graph distances with message passing neural networks. In *ICPR*, pages 2239–2244. IEEE Computer Society, 2018.

Kaspar Riesen. *Structural Pattern Recognition with Graph Edit Distance - Approximation Algorithms and Applications*. Advances in Computer Vision and Pattern Recognition. Springer, 2015. ISBN 978-3-319-27251-1. doi:10.1007/978-3-319-27252-8. URL https://doi.org/10.1007/978-3-319-27252-8.

Kaspar Riesen and Horst Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In Niels da Vitoria Lobo, Takis Kasparis, Fabio Roli, James T. Kwok, Michael Georgiopoulos, Georgios C. Anagnostopoulos, and Marco Loog, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, pages 287–297, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-89689-0.

Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vision Comput.*, 27(7):950–959, 2009. doi:10.1016/j.imavis.2008.04.004. URL https://doi.org/10.1016/j.imavis.2008.04.004.

Kaspar Riesen and Horst Bunke. *Graph Classification and Clustering Based on Vector Space Embedding*, volume 77 of *Series in Machine Perception and Artificial Intelligence*. WorldScientific, 2010a.

Kaspar Riesen and Horst Bunke. *Graph Classification and Clustering Based on Vector Space Embedding*, volume 77 of *Series in Machine Perception and Artificial Intelligence*. WorldScientific, 2010b. ISBN 978-981-4304-71-9. doi:10.1142/7731. URL https://doi.org/10.1142/7731.

Kaspar Riesen and Miquel Ferrer. Predicting the correctness of node assignments in bipartite graph matching. *Pattern Recognition Letters*, 69:8–14, 2016.

Kaspar Riesen, Stefan Fankhauser, and Horst Bunke. Speeding up graph edit distance computation with a bipartite heuristic. In Paolo Frasconi, Kristian Kersting, and Koji Tsuda, editors, *Mining and Learning with Graphs, MLG 2007, Firence, Italy, August 1-3, 2007, Proceedings*, 2007. URL http://mlg07.dsi.unifi.it/pdf/02_Riesen.pdf.

Volker Roth, Julian Laub, Motoaki Kawanabe, and Joachim M. Buhmann. Optimal cluster preserving embedding of nonmetric proximity data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25 (12):1540–1551, 2003. doi:10.1109/TPAMI.2003.1251147. URL http://dx.doi.org/10.1109/TPAMI.2003.1251147.

Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, July 1976. ISSN 0004-5411. doi:10.1145/321958.321975. URL http://doi.acm.org/10.1145/321958.321975.

Olfa Sammoud, Christine Solnon, and Khaled Ghédira. Ant algorithm for the graph matching problem. In *Proceedings of the 5th European Conference on Evolutionary Computation in Combinatorial Optimization*, EvoCOP'05, pages 213–223, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-25337-8, 978-3-540-25337-2. doi:10.1007/978-3-540-31996-2_20. URL http://dx.doi.org/10.1007/978-3-540-31996-2_20.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009. doi:10.1109/TNN.2008.2005605. URL https://doi.org/10.1109/TNN.2008.2005605.

Francesc Serratosa. Computation of graph edit distance: Reasoning about optimality and speed-up. *Image Vision Comput.*, 40:38–48, 2015. doi:10.1016/j.imavis.2015.06.005. URL https://doi.org/10.1016/j.imavis.2015.06.005.

Francesc Serratosa. Graph edit distance: Restrictions to be a metric. *Pattern Recognition*, 90: 250 – 256, 2019. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2019.01.043. URL http://www.sciencedirect.com/science/article/pii/S0031320319300639.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. doi:10.1017/CBO9781107298019.

L. G. Shapiro and R. M. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-3(5):504–519, Sept 1981. ISSN 0162-8828. doi:10.1109/TPAMI.1981.4767144.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, November 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2078187.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. doi:10.1038/nature16961.

Dinesh Singh and C. Krishna Mohan. Graph formulation of video activities for abnormal activity recognition. *Pattern Recognition*, 65:265 – 272, 2017. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2017.01.001. URL http://www.sciencedirect.com/science/article/pii/S0031320317300018.

Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific J. Math.*, 21(2):343–348, 1967. URL https://projecteuclid.org:443/euclid.pjm/1102992505.

Sébastien Sorlin, Christine Solnon, and Jean-Michel Jolion. A generic graph distance measure based on multivalent matchings. In Abraham Kandel, Horst Bunke, and Mark Last, editors, *Applied Graph Theory in Computer Vision and Pattern Recognition*, volume 52 of *Studies in Computational Intelligence*, pages 151–181. Springer, 2007. ISBN 978-3-540-68019-2. doi:10.1007/978-3-540-68020-8_6. URL https://doi.org/10.1007/978-3-540-68020-8_6.

Michael Stauffer, Thomas Tschachtli, Andreas Fischer, and Kaspar Riesen. A survey on applications of bipartite graph edit distance. In Foggia et al. [2017], pages 242–252. ISBN 978-3-319-58960-2. doi:10.1007/978-3-319-58961-9_22. URL https://doi.org/10.1007/978-3-319-58961-9_22.

Michael Stauffer, Andreas Fischer, and Kaspar Riesen. Keyword spotting in historical handwritten documents based on graph matching. *Pattern Recognition*, 81:240 – 253, 2018. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2018.04.001. URL http://www.sciencedirect.com/science/article/pii/S0031320318301274.

Paul Swoboda, Carsten Rother, Hassan Abu Alhaija, Dagmar Kainmuller, and Bogdan Savchynskyy. A study of lagrangean decompositions and dual ascent solvers for graph matching. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, pages 25–32, Cambridge, MA, USA, 2003. MIT Press. URL http://dl.acm.org/citation.cfm?id=2981345.2981349.

Damien Teney, Lingqiao Liu, and Anton van den Hengel. Graph-structured representations for visual question answering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

L. Torresani, V. Kolmogorov, and C. Rother. A dual decomposition approach to feature correspondence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(2):259–271, Feb 2013. ISSN 0162-8828. doi:10.1109/TPAMI.2012.105.

Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Feature correspondence via graph matching: Models and global optimization. In *Computer Vision - ECCV*, pages 596–609, 2008.

Wen-hsiang Tsai, Student Member, and King-sun Fu. Pattern Deformational Model and Bayes Error-Correcting Recognition System. *IEEE Transactions on Systems, Man, and Cybernetics*, 9:745–756, 1979.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 104–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi:10.1145/1015330.1015341. URL http://doi.acm.org/10.1145/1015330.1015341.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484, December 2005. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1046920.1088722.

Jeffrey K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175 – 179, 1991. ISSN 0020-0190. doi:https://doi.org/10.1016/0020-0190(91)90074-R. URL http://www.sciencedirect.com/science/article/pii/002001909190074R.

Julian R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976. doi:10.1145/321921.321925. URL http://doi.acm.org/10.1145/321921.321925.

L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984. ISSN 0001-0782. doi:10.1145/1968.1972. URL http://doi.acm.org/10.1145/1968.1972.

Ernest Valveny, Mathieu Delalandre, Romain Raveaux, and Bart Lamiroy. Report on the symbol recognition and spotting contest. In Young-Bin Kwon and Jean-Marc Ogier, editors, *Graphics Recognition. New Trends and Challenges - 9th International Workshop, GREC 2011, Seoul, Korea, September 15-16, 2011, Revised Selected Papers*, volume 7423 of *Lecture Notes in Computer Science*, pages 198–207. Springer, 2011. ISBN 978-3-642-36823-3. doi:10.1007/978-3-642-36824-0_19. URL https://doi.org/10.1007/978-3-642-36824-0_19.

Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

Mario Vento. A long trip in the charming world of graphs for pattern recognition. *Pattern Recognition*, 48(2):291 – 301, 2015. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2014.01.002. URL http://www.sciencedirect.com/science/article/pii/S0031320314000053.

Cédric Villani. *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer, 2009 edition, September 2008. ISBN 3540710493. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/3540710493.

Richard C. Wilson and Ping Zhu. A study of graph spectra for comparing graphs and trees. *Pattern Recognition*, 41(9):2833 – 2841, 2008. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2008.03.011. URL http://www.sciencedirect.com/science/article/pii/S0031320308000927.

Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, Jun 2015. ISSN 1436-4646. doi:10.1007/s10107-015-0892-3. URL https://doi.org/10.1007/s10107-015-0892-3.

Hongteng Xu, Dixin Luo, Hongyuan Zha, and Lawrence Carin. Gromov-wasserstein learning for graph matching and node embedding. *CoRR*, abs/1901.06003, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018.

Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics. ISBN 0-89871-313-7. URL http://dl.acm.org/citation.cfm?id=313559.313789.

Tianshu Yu and Ruisheng Wang. Scene parsing using graph matching on street-view data. *Computer Vision and Image Understanding*, 145:70 – 80, 2016. ISSN 1077-3142. doi:https://doi.org/10.1016/j.cviu.2016.01.004. URL http://www.sciencedirect.com/science/article/pii/S1077314216000175. Light Field for Computer Vision.

Andrei Zanfir and Cristian Sminchisescu. Deep learning of graph matching. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

M. Zaslavskiy, F. Bach, and J. Vert. A path following algorithm for the graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2227–2242, Dec 2009. ISSN 0162-8828. doi:10.1109/TPAMI.2008.245.

R. Zass and A. Shashua. Probabilistic graph and hypergraph matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008. doi:10.1109/CVPR.2008.4587500.

Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009. doi:10.14778/1687627.1687631. URL http://www.vldb.org/pvldb/2/vldb09-568.pdf.

Z. Zhang, Q. Shi, J. McAuley, W. Wei, Y. Zhang, and A. v. d. Hengel. Pairwise matching through max-weight bipartite belief propagation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1202–1210, June 2016a. doi:10.1109/CVPR.2016.135.

Zhen Zhang, Qinfeng Shi, Julian McAuley, Wei Wei, Yanning Zhang, and Anton van den Hengel. Pairwise matching through max-weight bipartite belief propagation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016b.

Qing Zhao, Stefan E. Karisch, Franz Rendl, and Henry Wolkowicz. Semidefinite programming relaxations for the quadratic assignment problem. *Journal of Combinatorial Optimization*, 2(1): 71–109, Mar 1998. ISSN 1573-2886. doi:10.1023/A:1009795911987. URL https://doi.org/10.1023/A:1009795911987.

Dengyong Zhou and Bernhard Schölkopf. Regularization on discrete spaces. In Walter G. Kropatsch, Robert Sablatnig, and Allan Hanbury, editors, *Pattern Recognition*, pages 361–368, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31942-9.

F. Zhou and F. De la Torre. Factorized graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(9):1774–1789, Sept 2016. ISSN 0162-8828. doi:10.1109/TPAMI.2015.2501802.

Shlomo Zilberstein and Stuart J. Russell. Approximate reasoning using anytime algorithms. In S. Natarajan, editor, *Imprecise and Approximate Computation*. 1995.

Part III

Appendix

Appendix A

More graph matching problems

A.1 Graph isomorphism

The mapping, or matching, between the vertices of the two graphs must be edge-preserving in the sense that if two vertices in the first graph are linked by an edge, they are mapped to two vertices in the second graph that are linked by an edge as well. This condition must be held in both directions, and the mapping must be bijective. That is, a one-to-one correspondence must be found between each vertex of the first graph and each vertex of the second graph. When graphs are attributed, attributes have to be identical. More formally, when comparing two graphs $G_1 = (V_1,E_1,\mu_1,\zeta_1)$ and $G_2 = (V_2,E_2,\mu_2,\zeta_2)$, we are looking for a bijective function $f : V_1 \to V_2$ which maps each vertex $u_i \in V_1$ onto a vertex $v_k \in V_2$ such that certain conditions are fulfilled:

Definition 23. *Graph isomorphism*
A bijective function $f : V_1 \to V_2$ is a graph isomorphism from G_1 to G_2 if:

1. $\forall u_i \in V_1,\ \mu_1(u_i) = \mu_2(f(u_i))$
2. $\forall u_i, u_j \in V_1, (u_i, u_j) \in E_1 \Leftrightarrow (f(u_i), f(u_j)) \in E_2$
3. $\forall (u_i, u_j) \in E_1,\ \zeta_1((u_i, u_j)) = \zeta_2((f(u_i), f(u_j)))$

In the dissertation, the term *source* graph refers to graph G_1 while *target* graph refers to G_2. Graph isomorphism is one of the problems for which it has not yet been demonstrated if it belongs to $\mathcal{NP}$-complete or not. However, there is still no algorithm that can solve the problem in polynomial time. Yet, readers who are aware of the recent rise of graph isomorphism might have heard about the claim of L. Babai in [Babai, 2015] of solving graph isomorphism in quasipolynomial time.

A.2 Monomorphism

Monomorphism, also known as partial subgraph isomorphism, is a light form of induced subgraph isomorphism. It also drops the condition that the mapping should be edge-preserving in both directions. It requires that each vertex of the source graph is mapped to a distinct vertex of the target graph, and each edge of the source graph has a corresponding edge in the target graph. However, the target graph may have both extra vertices and extra edges.

The subgraph monomorphism problem between a pattern graph G_1 and a target graph G_2 is defined by:

Definition 24. *Monomorphism*
An injective function $f : V_1 \to V_2$ is a subgraph isomorphism from G_1 to G_2 if:

1. $\forall u_i \in V_1,\ \mu_1(u_i) = \mu_2(f(u_i))$
2. $\forall u_i, u_j \in V_1, (u_i, u_j) \in E_1 \Rightarrow (f(u_i), f(u_j)) \in E_2$
3. $\forall (u_i, u_j) \in E_1,\ \zeta_1((u_i, u_j)) = \zeta_2((f(u_i), f(u_j)))$

A.3 Substitution-Tolerant Subgraph Isomorphism

Substitution-Tolerant Subgraph Isomorphism [Bodic et al., 2012] aims at finding a subgraph isomorphism of a pattern graph G_s in a target graph G. This isomorphism only considers label substitutions and forbids vertex and edge insertion in G. This kind of subgraph isomorphism is often needed in PR problems when graphs are attributed with real values and no exact GM can be found between attributes due to noise. A subgraph isomorphism is said to be substitution-tolerant when the mapping does not affect the topology. That is, each vertex and each edge of the pattern graph has a one-to-one mapping into the target graph, however, two vertices and/or edges can be matched (or substituted) even if their attributes are not similar. A substitution-tolerant mapping is generally needed when no exact mapping between vertex and/or edge attributes can be found, but when the mapping can be associated to penalty cost. For example, this case occurs when vertex and edge attributes are numerical values (scalar or vectorial) resulting from a feature extraction step as often in pattern analysis.

Definition 25. *Substitution-Tolerant Subgraph Isomorphism*
An injective function $f : V_1 \to V_2$ is a subgraph isomorphism of $G_1 = (V_1,E_1,L_{V_1},L_{E_1},\mu_1,\zeta_1)$ and $G_2 = (V_2,E_2,L_{V_2},L_{E_2},\mu_2,\zeta_2)$ if the following conditions are satisfied:

1. $\forall u_i \in V_1,\ \mu_1(u_i) \approx \mu_2(f(u_i))$
2. $\forall u_i, u_j \in V_1, (u_i, u_j) \in E_1 \Leftrightarrow (f(u_i), f(u_j)) \in E_2$
3. $\forall (u_i, u_j) \in E_1, \zeta_1((u_i, u_j)) \approx \zeta_2((f(u_i), f(u_j)))$

In PR applications, where vertices and edges are labeled with measures which may be affected by noise, a substitution-tolerant formulation which allows differences between attributes of mapped vertices and edges is mandatory. However, these differences are associated to costs where the objective is to find the mapping corresponding to the minimal global cost, if one exists. i.e., $\mu_1(u_i) \approx \mu_2(v_k)$ and $\zeta_1(e(u_i, u_j)) \approx \zeta_2(e(v_k, v_z))$.

A.4 Multivalent Matching

All the aforementioned matching problems, whether exact or error-tolerant ones, belong to the univalent family in the sense of allowing one vertex or one edge of one graph to be substituted with one and only one vertex or edge in the other graph.

In many real-world applications, comparing patterns described at different granularity levels is of great interest. For instance, in the field of image analysis, an over-segmentation of some images might occur whereas an under-estimation occurs in some other images resulting in allowing several regions of one image to be correspondent, or related to, a single region of another image. Based on this fact, multivalent matching problem emerged to be one of the interesting problems in graph

theory [Sorlin et al., 2007]. Multivalent matching drops the condition that vertices in the source graph are to be mapped to distinct vertices of the target graph. Thus, in multivalent matching, vertex in the first graph can be matched with an empty set of vertices, one vertex or even multiple vertices in the other graph. This matching problem is also called relational matching since GM is no longer a function but rather a relation $m \subseteq V_1 \times V_2$. The objective of this kind of matching is to minimize the number of split vertices (i.e., vertices that are matched with more than one vertex).

Mathematically, the relation m associating a vertex of one graph to a set of vertices of the other graph can be defined as follows:

Definition 26. *Multivalent Matching*
A relation $m \subseteq V_1 \times V_2$ is a multivalent matching from G_1 to G_2 if:

1. $\forall u_i \in V_1,\ m(u_i) \approx \{v_k \in V_2 | (u_i, v_k) \in m\}$
2. $\forall v_k \in V_2,\ m(v_k) \approx \{u_i \in V_1 | (u_i, v_k) \in m\}$

where $m(v_*)$ denotes the set of vertices that are associated with a vertex v_* by the relation m.

Appendix B

Machine learning theory

The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as *classifying* the data into different categories or predicting continuous variables, then the task is called *regression*. A set of examples $\{x_1, \cdot, x_M\}$ along with their corresponding target (categories or real value) $\{t_1, \cdot, t_M\}$ is called a training set. Targets are also called ground-truth or predictions in machine learning. The training set is used to tune the parameters of an adaptive model. The learning algorithm is concerned by the adaptation of the parameters of model. A model or a learning algorithm can have meta-parameters that are not learned by the learning algorithm. Applications in which the training data comprises examples along with their corresponding target are known as *supervised learning* problems. In other pattern recognition problems, the training data consists of a set of input x without any corresponding target values. The goal in such *unsupervised learning* problems may be to discover groups of similar examples within the data then it is called clustering or to determine the distribution of data. Finally, the technique of *reinforcement learning* is concerned with the problem of finding suitable actions to take in a given situation in order to maximize a reward. Here the learning algorithm is not given examples and neither the target, but must instead discover them by a process of trial and error.

The ability to predict or classify correctly new examples that differ from those used for training is known as *generalization*. In practical applications, the variability of the input vectors will be such that the training data can comprise only a tiny fraction of all possible examples, and so generalization is a central goal in pattern recognition.

The overfitting problem occurs when the model fails to generalize on new unseen data. This means that the accuracy of the prediction on the training data set are very high while it performs poorly on a new data set. The reverse problem exists also, the inability of a learning algorithm to reach an acceptable level of accuracy on the training data set. To be able to measure these problems, usually, an entire data set is split into three parts :

- Train data set: It is the data set used directly by the learning algorithm. On this data set, parameters are updated in order to achieve a goal.
- Validation data set: on this data set, the learning process does not occur. This means that parameters are fixed and will never be updated using this data set. However, this data set is still useful because it helps to validate the model on pieces of data unseen during the learning process. Validation data set is meaningful to adapt the meta-parameters of the learning algorithm.
- Test data set: After the learning process has been completed, the test data set comes into play. Its objective is to test the model on completely new and unseen data. Parameters and

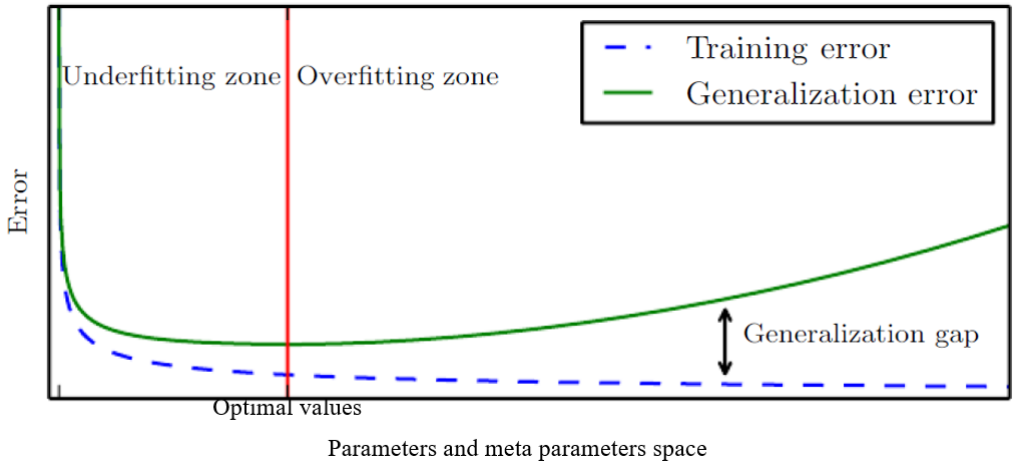


Figure B.1: Selection of the best parameters and meta parameters.

meta parameters are fixed. No learning mechanism occurs on this data set.

On these three data sets an error measure of the model can be computed (i.e. a classification error). The error measure can help to identify the best parameter and meta-parameter values as pictured out in Figure B.1.

Error measures depend on the goal to be achieve by the learning algorithm. The learning algorithm is often stated as a minimization problem where the objective function represent the goal to be reached. The objective function is also called loss function or cost function in machine learning.

B.1 The basics of losses

Many loss functions have been designed. We take a closer look to two loss functions. The cross entropy that is well suited for classification tasks and the least square errors that is good for regression task.

B.1.1 Cross-entropy loss

This paragraph describes how minimizing the cross-entropy is related with maximizing the likelihood of the model according to the training set. This consideration was first explained by Vapnik in [Vapnik, 1998]. When we develop a model for classification, we aim to map the model's inputs to targets. The targets can be encoded by integers or by an encoding called one-hot vector. For example, if we're interested in determining whether an image is best described as a landscape or as a house or car. The target of the first sample t_1 can be represented as $t_1 \in [1, 2, 3]$ or $t_1 \in \{0, 1\}^3$. If an image is a house then $t_1 = [1, 0, 0]$. A prediction made by the classifier could be $\hat{t}_1 = [0.2, 0.3, 0.5]$. The notation $\hat{t}_1^1$ refers to the first value of the vector $\hat{t}_1$.

In cross-entropy there is entropy. The entropy of the discrete random variable t_1 is defined as :

$$H(t_1) = \sum_i Pr(t_1^i) \log_2 \frac{1}{Pr(t_1^i)} = -\sum_i Pr(t_1^i) \log_2 Pr(t_1^i)$$

If we assume that t_1^i is already a probability then :

$$H(t_1) = \sum_i t_1^i \log_2 \frac{1}{t_1^i} = -\sum_i t_1^i \log_2 t_1^i$$

The entropy measures the number of bits needed to encode the classes of t_1. In contrast, the cross entropy is the number of bits we need if we encode classes from t_1 using $\hat{t}_1$.

$$H(t_1, \hat{t}_1) = \sum_i t_1^i \log_2 \frac{1}{\hat{t}_1^i} = -\sum_i t_1^i \log_2 \hat{t}_1^i$$

Cross-entropy is always larger than entropy; encoding classes according to the wrong distribution $\hat{t}_1$ will always make us use more bits. The only exception is the trivial case where t_1 and $\hat{t}_1$ are equal, and in this case entropy and cross entropy are equal. The Kullback–Leibler (KL) divergence from $\hat{t}_1$ to t_1 is simply the difference between cross-entropy and entropy:

$$KL(t_1 \mid\mid \hat{t}_1) = \sum_i t_1^i \log \frac{1}{\hat{t}_1^i} - \sum_i t_1^i \log \frac{1}{t_1^i} = \sum_i t_1^i \log \frac{t_1^i}{\hat{t_1}^i} \tag{B.1}$$

It measures the number of extra bits we'll need on average if we encode classes from t_1 according to $\hat{t}_1$.

It's never negative, and it's 0 only when t_1 and $\hat{t}_1$ are the same.

Note that minimizing cross entropy is the same as minimizing the KL divergence from $\hat{t}_1$ to t_1.

Now we can use the cross entropy over all training examples as our loss. In particular, if we let n index training examples, the overall loss would be

$$H(t, \hat{t}) = \sum_{n=1}^{M} H(t_n, \hat{t}_n)$$

We have define the cross entropy, and it seems quite relevant but is there any reason?

B.1.1.1 The cross entropy, a probabilistic reasoning

But let's look at another approach. What if we want our objective function to be a direct measure of our model's predictive power, at least with respect to our training data? One common approach is to tune our parameters so that the likelihood of our data under the model is maximized.

$$\max Pr(t|\hat{t})$$

We usually assume that our samples are independent and identically distributed (iid), each measure is independent from the other, so, the likelihood over all of our examples decomposes into a product over the likelihoods of individual examples:

$$\max Pr(t|\hat{t}) = \Pi_{n=1}^{M} \max \ Pr(t_n|\hat{t}_n)$$

Going back to the original example, if the first training image is of a landscape, then $t_1 = (1.0, 0.0, 0.0)^T$, which tells us that the likelihood $Pr(t_1, \hat{t}_1)$ is just the first entry of $\hat{t}_1 = (0.2, 0.3, 0.5)^T$, which is $\hat{t}_1^1 = 0.2$.

Let's play a bit with the likelihood expression above.

First, since the logarithm is monotonic, we know that maximizing the likelihood is equivalent to maximizing the log likelihood, which is in turn equivalent to minimizing the negative log likelihood:

$$-\log Pr(t|\hat{t}) = -\sum_{n=1}^{M} \log Pr(t_n|\hat{t}_n)$$

But from our discussion above, we also know that the log likelihood of t_n is just the log of a particular entry of $\hat{t}_n$. In fact, it's the entry i which satisfies $t_n^i = 1$. We can therefore rewrite the log likelihood for the n-th training example in the following way:

$$\log Pr(t_n|\hat{t}_n) = \sum_{i=1}^{3} t_n^i \log \hat{t}_n^i$$

which gives us an overall negative log likelihood of

$$-\log Pr(t|\hat{t}) = -\sum_{n=1}^{M}\sum_{i=1}^{3} t_n^i \log \hat{t}_n^i$$

This is precisely cross entropy, summed over all training examples:

$$-\log Pr(t|\hat{t}) = \sum_{n=1}^{M} \big[-\sum_{i=1}^{3} t_n^i \log \hat{t}_n^i \big] = \sum_{n=1}^{M} H(t_n, \hat{t}_n)$$

In the context of a discriminative model for probabilistic classification, minimizing the cross entropy is equivalent to maximize the likelihood of $Pr(t|\hat{t})$ without any assumption on the distributions t and $\hat{t}$.

B.1.2 Mean square error loss

We shall see that the least squares approach to finding the model parameters represents a specific case of maximum likelihood. We now use the training data $\{x, t\}$ where the encoding of t_n is not the one-hot vector but an integer $t_n \in \mathbb{N}$. The least square error can be defined as follows :

$$MSE = \frac{1}{M}\sum_{n=1}^{M} \{\hat{t}_n - t_n\}^2 = \frac{1}{M}||\hat{t}_n - t_n||_2^2$$

We now use the training data to determine the values of the unknown parameters w of our predictor $f(x, w)$ by maximum likelihood. For this purpose, we shall assume that, given the value of x, the corresponding value of t has a Gaussian distribution with a mean equal to the value $\hat{t}_n = f(x_n, w)$.

$$Pr(t|x, w, \sigma) = Pr(t|\hat{t}, \sigma) = \mathcal{N}(t|\hat{t}_n, \sigma)$$

σ is the variance of the distribution. We just recall the equation of a Gaussian distribution.

$$Pr(x|\mu, \sigma) = \mathcal{N}(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

If the data are assumed to be drawn independently from the distribution, then the likelihood function is given by

$$Pr(t|\hat{t}, \sigma) = \Pi_{n=1}^{M} \mathcal{N}(t_n|\hat{t}_n, \sigma)$$

As we did in the case of the cross-entropy, it is convenient to maximize the logarithm of the likelihood function.

$$\log Pr(t|\hat{t}, \sigma) = -\frac{1}{2\sigma}\sum_{n=1}^{M} \{\hat{t}_n - t_n\}^2 - \frac{M}{2}\log\sigma - \frac{M}{2}\log(2\pi) \tag{B.2}$$

$$\log Pr(t|x, w, \sigma) = -\frac{1}{2\sigma}\sum_{n=1}^{M} \{f(x_n, w) - t_n\}^2 - \frac{M}{2}\log\sigma - \frac{M}{2}\log(2\pi) \tag{B.3}$$

Consider first the determination of the maximum likelihood solution which will be denoted by w^*. These are determined by maximizing EquationB.3 with respect to w. For this purpose, we can omit the last two terms on the right-hand side of Equation B.3 because they do not depend on w.

Also, we note that scaling the log likelihood by a positive constant coefficient does not alter the location of the maximum with respect to w, and so we can replace the coefficient $\frac{1}{2\sigma}$ with $\frac{1}{M}$. Finally, instead of maximizing the log likelihood, we can equivalently minimize the negative log likelihood.

$$\frac{\partial - logPr(t|x, w, \sigma)}{\partial w} = -\frac{1}{\sigma}\sum_{n=1}^{M}\{f(x_n, w) - t_n\} = 0 \tag{B.4}$$

We therefore see that maximizing likelihood is equivalent, so far as determining w is concerned, to minimizing the sum-of-squares error function defined by Equation B.4.

Thus the sum-of-squares error function has arisen as a consequence of maximizing likelihood under the assumption of a Gaussian noise distribution.

B.1.3 Regularization of loss functions

Regularization is a common feature of machine learning technique to deal with generalization problem. One technique that is often used to control the over-fitting phenomenon in such cases is that of regularization, which involves adding a penalty term to the error function in order to discourage the coefficients from reaching large values. The simplest such penalty term takes the form of a sum of squares of all of the coefficients, leading to a modified error function of the form:

$$Loss = \frac{1}{M}\sum_{n=1}^{M}\{f(x_n, w) - t_n\}^2 + \frac{\lambda}{2}||w||_2^2$$

where $||w||_2^2$ and the coefficient λ governs the relative importance of the regularization term compared with the sum-of-squares error term. The $L2$ norm regularization can be explained by a Bayesian approach.

Therefore we introduce a prior distribution over the coefficients w. Let us consider a Gaussian distribution of the form

$$Pr(w|\alpha) = \mathcal{N}(w|0, \alpha) = \exp\left(-\frac{\alpha}{2}w^T w\right)$$

where *alpha* is the variance of the distribution. Variables such as α, which control the distribution of model parameters, are called hyperparameters. Using Bayes' theorem, the posterior distribution for w is proportional to the product of the prior distribution and the likelihood function

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior}$$

$$Pr(w|x, t, \alpha, \sigma) \propto Pr(t|\hat{t}, \sigma)Pr(w|\alpha)$$

We can now determine w by finding the most probable value of w given the data, in other words by maximizing the posterior distribution. This technique is called *Maximum A Posteriori*, or simply MAP. Taking the negative logarithm, we find that the maximum of the posterior is given by the minimum of

$$\frac{1}{2\sigma}\sum_{n=1}^{M}\{f(x_n, w) - t_n\}^2 + \frac{\alpha}{2}w^T w$$

We see that maximizing the posterior distribution is equivalent to minimizing the regularized sum-of-squares error function with a regularization parameter given by $\lambda = \sigma\alpha$. This statement can be found in [Bishop, 2006]. Many other regularization terms can be found in the literature such as $L1$ norm or the pseudo norm $L0$. Classically, the cost function in a learning problem can be written as :

Cost Function= Loss (i.e. MSE or cross entropy) + Regularization term

B.2 Probably Approximately Correct (PAC)

This has its origins with Valiant [1984] who formulated the probably approximately correct, or PAC, learning framework. The goal of the PAC framework is to understand how large a data set needs to be in order to give good generalization. It also gives bounds for the computational cost of learning.

Suppose that a data set $\mathcal{D}$ of size M is drawn from some joint distribution $Pr(x, t)$ where x is the input variable and t represents the class label, and that we restrict attention to 'noise free' situations in which the class labels are determined by some (unknown) deterministic function $t = f(x)$. In PAC learning we say that a function $g(x, D)$, drawn from a space $\mathcal{F}$ of such functions on the basis of the training set $\mathcal{D}$, has good generalization if its expected error rate is below some pre-specified threshold ϵ, so that

$$\mathbb{E}_{x,t}[I(g(x; \mathcal{D}) = t)] < \epsilon$$

where $I(.)$ is the indicator function, and the expectation is with respect to the distribution $Pr(x, t)$. PAC learning aims to provide bounds on the minimum size M of data set needed to meet this criterion. A key quantity in PAC learning is the Vapnik-Chervonenkis dimension, or VC dimension, which provides a measure of the complexity of a space of functions. The bounds derived within the PAC framework are often described as worst case, because they apply to any choice for the distribution $Pr(x, t)$, so long as both the training and the test examples are drawn (independently) from the same distribution, and for any choice for the function $g(x)$ so long as it belongs to $\mathcal{F}$. The PAC bounds are very conservative, in other words they strongly over-estimate the size of data sets required to achieve a given generalization performance.

In real-world applications of machine learning, we deal with distributions that have significant regularity, for example in which large regions of input space carry the same class label.

B.3 Structured prediction

Generalize classification/regression methods to deal with structured outputs and/or with multiple, interdependent outputs. Outputs are either

- Structured objects such as sequences, strings, trees, etc.
- Variables that are interdependent (e.g. dependencies modeled by probabilistic graphical models)

One of the easiest ways to understand algorithms for general structured prediction is the structured perceptron of Collins [Collins, 2002]. This algorithm combines the perceptron algorithm for learning linear regressor with an inference algorithm (classically the Viterbi algorithm when used on sequence data) and can be described abstractly as follows. First define a "joint feature function" $\Phi(x, y)$ that maps a training sample x and a feasible solution y to a vector of length d (x and y may have any structure; d is problem-dependent, but must be fixed for each model). Let GEN be a function that generates a set of feasible solutions and α is the learning rate.

- Let β be a weight vector of length n.
- For a pre-determined number of iterations:
  - For each sample x in the training set with true output t:
    - Find a feasible solution $y^* = \underset{y \in GEN(x)}{arg\,min}\ (\beta^T \phi(x, y))$
    - Update β, from y^* to t: $\beta = \beta + \alpha(-\Phi(x, y^*) + \Phi(x, t))$

This simple paradigm can be extended to the large margin framework [Tsochantaridis et al., 2005]:

$$\min_\beta \frac{1}{2}||\beta||_2^2 \tag{B.5}$$

$$\text{Such that:} \tag{B.6}$$

$$\beta.\Phi(x, y^*) - \beta.\Phi(x, t) \geq 1 \tag{B.7}$$

$$\forall (x, t) \in \mathcal{D} \text{ and } y^* \in \mathcal{Y} \tag{B.8}$$

There are an exponential number of constraints for each input.

Appendix C

Graph neural networks

C.1 History

One of the key reasons for the success of deep neural networks is their ability to leverage statistical properties of the data such as stationarity and compositionality through local statistics.

So, a lot of attention has been devoted to the generalization of neural network models to structured datasets.

In the last five years, a number of papers re-visited this problem of generalizing neural networks to work on arbitrarily structured graphs [Bruna et al., 2013, Henaff et al., 2015, Duvenaud et al., 2015, Li et al., 2015, Kipf and Welling, 2016], some of them now achieving very promising results in domains that have previously been dominated by, e.g., kernel-based methods, graph-based regularization techniques. These former methods were based on a two-step pipeline:

1. Get embedding for every node or graph
2. Train a classifier on node/graph embedding

At the opposite, Graph Neural Networks (GNN) perform an end-to-end learning including feature extraction and classification.

A time-line about graph neural networks is depicted on Figure C.1.

C.2 The basics of artificial neural networks

An artificial neural networks is a model of data. It is composed of layers organized hierarchically. Each layer is composed of a set artificial neurons. Inside a layer, the neurons are not structured and do not communicate. Between layers, neurons can communicate by sending their output to the input of the next layer.

Let $x \in \mathbb{R}^{1\times m}$ be a vector considered as an input data. It is also called the input signal.

Let a layer be defined as :

$$H^{(l+1)} = f(H^{(l)})$$
$$H^{(l+1)} = \sigma(H^{(l)}W^{(l)}) \quad \forall l > 0$$

$W^{(l)} \in \mathbb{R}^{m_l \times m_{l+1}}$ is a matrix of trainable parameters. m_l is the number of neurons of the layer l. For the layer 0, $H^{(0)} = x$. Layer $l+1$ produces a vector $H^{(l+1)} \in \mathbb{R}^{1\times m_{l+1}}$. Finally, σ is a non linear function. This neural network is considered as a model where parameters can be learned.

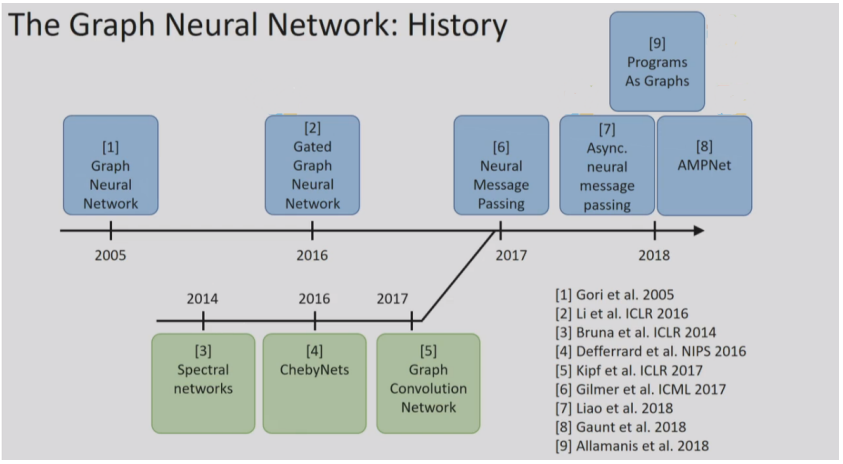


Figure C.1: A time line about graph neural networks.

This model is also denoted as a *"dense" layer* or *"Fully Connected (FC)" layer* or a *"MuLtilayer Perceptron"* (MLP). The question is how to generalize this artificial neural networks to graphs? What to do when the input is a graph?

C.3 The basics of graph neural networks

C.3.1 Definitions

Assume we have a graph G:

- V is the vertex set.
- E is the edge set.
- A is the adjacency matrix (assume binary). $A \in \{0,1\}^{|V|\times|V|}$
- F $\in \mathbb{R}^{|V|\times m}$ is a matrix of node features.
  - Categorical attributes, text, image data
  - Node degrees, clustering coefficients, etc.
  - Indicator vectors (i.e., one-hot encoding of each node)

In the literature, the graph structure is also called "domain structure" while features are also named "data on a domain" [Bronstein et al., 2016]. Some papers of the literature makes a clear distinction between fixed domain structure [Henaff et al., 2015, Defferrard et al., 2016] or variable domain structure [Atwood and Towsley, 2015, Boscaini et al., 2016]. The last concept means that the input graphs can have different sizes and structures. The fixed domain structure problem often appear where the input is a single (large) graph while the variable domain structure is likely to appear when the learning set is composed of many (small) graphs. In [Scarselli et al., 2009], both paradigms are merged into a single one. The learning set composed of many graphs can be combined into a unique disconnected graph, and, therefore, one might think of the learning set as a pair composed of a single graph and the targets. It is worth mentioning that this compact definition is not only useful for its simplicity, but that it also captures directly the modelling power of graphs.

C.3.2 Intuition

The key idea is to generate node embeddings based on local neighborhoods. The intuition is to aggregate node information from their neighbors using neural networks. Nodes have embeddings at each layer and the neural network can be arbitrary depth. "layer-0" embedding of node u is its input feature, i.e. F_u. A GNN produces a node-level output Z (an $|V| \times p$ feature matrix, where p is the number of output features per node). Graph-level outputs can be modeled by introducing some form of pooling operation (see, e.g. [Duvenaud et al., 2015]).

A graph is processed by a set of units, each one corresponding to a node of the graph, which are linked according to the input graph connectivity. The graph structure is shared over layers.

Every graph neural network layer can then be written as a non-linear function:

$$H^{(l+1)} = f(H^{(l)}, A)$$

with $H^{(0)} = F$ and $H^{(L)} = Z$, L being the number of layers. The specific models then differ only in how $f(.,.)$ is chosen and parameterized.

A GNN layer receives as input a signal $H^{(l)} \in \mathbb{R}^{|V| \times m_l}$ and produces $H^{(l+1)} \in \mathbb{R}^{|V| \times m_{l+1}}$. This output can be fed into any loss function. The training algorithm is based on stochastic gradient descent to learn the aggregation parameters.

C.3.3 A simple example for $f(.,.)$:

As an example, let's consider the following very simple form of a layer-wise propagation rule:

$$f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right)$$

where $W^{(l)} \in \mathbb{R}^{m_l \times m_{l+1}}$ is a weight matrix for the l-th neural network layer. m_l is indexed by l because it depends on the number of parameters in the layer l. Theses values are hyperparameters of the model except for $H^{(0)} = F$. $\sigma(.)$ is a non-linear activation function like the ReLU. Note that $\sigma(.)$ is a element-wise non-linearity operating on a matrix. But first, let us address two limitations of this simple model: multiplication with A means that, for every node, we sum up all the feature vectors of all neighboring nodes but not the node itself (unless there are self-loops in the graph). This can be "fixed" by enforcing self-loops in the graph: we simply add the identity matrix to A.

The second major limitation is that A is typically not normalized and therefore the multiplication with A will completely change the scale of the feature vectors. Normalizing A such that all rows sum to one, i.e. $D^{-1}A$, where D is the diagonal node degree matrix, gets rid of this problem. Multiplying the input with $D^{-1}A$ now corresponds to taking the average of neighboring node features from the layer l. It is also called in the literature "Average neighbor messages" passing average node feature from one layer to another.

$$f(H^{(l)}, A) = \sigma\left(D^{-1}AH^{(l)}W^{(l)}\right)$$

After K-layers of neighborhood aggregation (compositionality), the network outputs embeddings for each node. The GNN can be seen as an *encoder* that maps nodes to vector embeddings:

$$ENC : G \to \mathbb{R}^{|V| \times p}$$

Finally, this model can be understood as many ($|V|$) dense layers (one for each node) working together thanks to an aggregation operator (see Figure C.2). Inside a given layer, the parameters $W^{(l)}$ are shared. So, the number of parameters does not depend on the number of nodes ($|V|$). Neighborhood aggregation can be viewed as a center-surround filter. It is mathematically

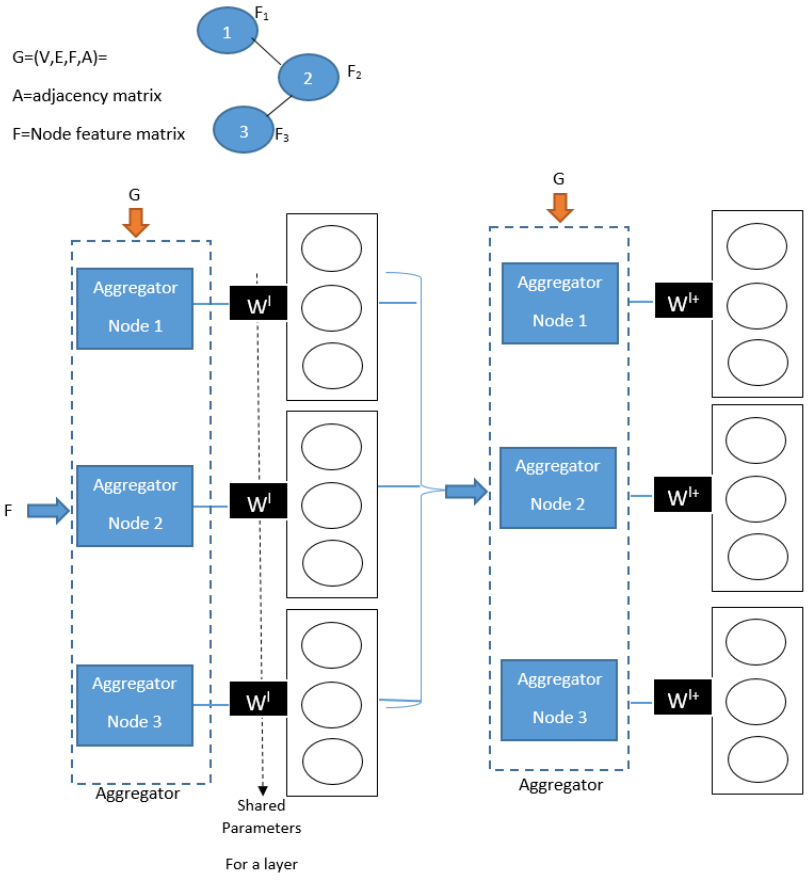


Figure C.2: A general overview of a GNN.

related to spectral graph convolutions (see [Bronstein et al., 2016]). Therefore, the GNN are also called Graph Convolutional Neural Network (GCNN). Thanks to the parameter sharing, the input graphs can have different sizes and structures. Such a model represents a *prior* on the data. Key assumptions are that graph-structured data is locally stationary and expressible by compositions (compositionality) and self-similar across the domain.

C.3.4 More complex message aggregations

C.3.4.1 Weighted aggregations

More complex functions have been applied in the literature. The key idea is to do more than averaging the features from a neighborhood. In [Kipf and Welling, 2016], a better (symetric) normalization of the adjacency matrix is proposed i.e. $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ (as this no longer amounts to mere averaging of neighboring nodes). A per-neighbor normalization is performed instead of simple average, normalization varies across neighbors.

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

with $\hat{A} = A + I$, where I is the identity matrix and $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$. The complexity of this model is $O(|E|)$ time complexity overall. However, it is not suited for regular

graphs (see [1] for more details).

More operations have been investigated in the literature [Nowak et al., 2017]. A complete family of operations can be used :

- $\mathbf{I} \in \mathbb{R}^{|V|\times|V|}$. This identity operator does not consider the structure of the graph and neither provide any aggregation. Used alone this operator makes the GNN a composition of $|V|$ MLP completly independent. One MLP for each node feature vector.
- $A \in \mathbb{R}^{|V|\times|V|}$. The adjacency operator gather information on the node neighborhood (1 hop).
- $D \in \mathbb{R}^{|V|\times|V|}$. $D = diag(A\mathbf{1}$. This degree operator gather information on the node degree. D is node degree matrix (a diagonal matrix).
- $A_j \in \mathbb{R}^{|V|\times|V|}$. $A_j = min(1, A^{2^j})$. It encodes 2^j-hop neighborhoods of each node, and allow us to aggregate local information at different scales, which is useful in regular graphs.
- U $\in 1^{|V|\times|V|}$. U is matrix filled with ones. This average operator, which allows to broadcast information globally at each layer, thus giving the GNN the ability to recover average degrees, or more generally moments of local graph properties.

By denoting $\mathcal{A} = \{\mathbf{1}, D, A, A_1, \cdots, A_J, U\}$, a GNN layer is defined as :

$$f(H^{(l)}, \mathcal{A}) = \sigma\left(\sum_{B\in\mathcal{A}} BH^{(l)}W_B^{(l)}\right)$$

$\Omega = \{W_1^{(l)}, \cdots, W_{|\mathcal{A}|}^{(l)}\}$, $W_B^{(l)} \in \mathbb{R}^{m_{(l)}\times m_{(l+1)}}$ are trainable parameters. All the nodes share the same operators but it is not mandatory.

C.3.4.2 Mean, max and neural network aggregations

Key distinctions are in how different approaches aggregate messages. So far, proposals have aggregated the neighbor messages by taking their (weighted) average, but is it possible to do better? In [Hamilton et al., 2017], a GNN called GraphSAGE is proposed. The aggregation of neighbors information is more complex. The very general scheme of aggregation can written thanks to the function AGG:

$$H^{(l+1)} = \sigma\left(AGG(H^{(l)})W^{(l)}\right)$$

Let us define $\mathcal{N}(u)$ is the set of nodes in the 1-hop neighborhood of node u.

- mean : $AGG_u = \frac{1}{|\mathcal{N}(u)|}\sum_{v\in\mathcal{N}(u)} H_v^{(l)} \quad \forall u \in V \implies AGG = D^{-1}AH^{(l)}$.
- max : $AGG_u = max(\{H_v^{(l)}, \quad \forall v \in \mathcal{N}(u)\}) \quad \forall u \in V$. Transform neighbor vectors into a matrix and apply a max pooling element-wise.
- LSTM : $AGG_u = LSTM([H_v^{(l)}, \quad \forall v \in \pi(\mathcal{N}(u))]) \quad \forall u \in V$. Where π is a random permutation. The idea is to provide to the LSTM a sequence composed of neighbor embeddings. So the input sequence is composed of vectors. The sequence is randomly permuted by the function π.

[1] https://www.inference.vc/how-powerful-are-graph-convolutions-review-of-kipf-welling-2016-2/

C.3.4.3 A word on permutation invariance

In the case of permutation invariant graphs, it is assumed that the learning task is independent of the order of neighbors. To generalize to unseen nodes or graphs at the decision stage, it is required that the (aggregator AGG) function acting on the neighbors must be invariant to the orders of neighbors under any random permutation (pi). In [Liu et al., 2018], a theorem on sufficiency and necessity conditions of AGG functions is stated. It is trivial to check that the LSTM aggregator in GraphSAGE is not a valid function under this condition, even though it could be possibly an appropriate aggregator function in temporal graphs.

C.4 Applications and losses

C.4.1 Unsupervised

It is possible to train in an unsupervised manner using only the graph structure and the features for instance. Let us recall that Z is the output the GNN. The graph factorization problem is the problem of predicting if two nodes are linked or not. It can then be defined as follows:

$$l = \sum_{(u_i,u_j)\in\mathcal{D}} (Z_i^T Z_j - A_{i,j})^2$$

Where $Z_i^T Z_j$ is a similarity measure between two nodes embeddings and A adjacency matrix. $\mathcal{D}$ is a database containing pair of nodes from different graphs or a single graph. However, the pair u_i and u_j must come from the same graph. Generally speaking, instead of $A_{i,j}$, any similarity function ($S : V \times V \to \mathbb{R}$) between u_i and u_j can be used. $Z_i^T Z_j$ is also called a *decoder* ($DEC(Z_i, Z_j)$). More complex decoder can be built. Like the matrix factorization approaches described above, DeepWalk [Perozzi et al., 2014] and node2vec [Grover and Leskovec, 2016] rely on embedding and use a decoder based on the inner product. However, instead of trying to decode a deterministic node similarity measure, these approaches optimize embeddings to encode the statistics of random walks. The basic idea behind these approaches is to learn embeddings so that :

$$DEC(Z_i, Z_j) = \frac{\exp(Z_i^T Z_j)}{\sum_{u_k\in V} \exp(Z_i^T Z_k)} = p_G(u_j|u_i)$$

Where $p_G(u_j|u_i)$ is the probability of visiting u_j on a length-k random walk starting at u_i.

$$l = \sum_{(u_i,u_j)\in\mathcal{D}} -S(u_i, u_j) \log(DEC(Z_i, Z_j))$$

Where $S(u_i, u_j) = ((D^{-1}A)^k)_{i,j}$ is the value of the random walk of length k.

C.4.2 Supervised

This alternative aims at directly training the model for a supervised task (e.g node classification, node regression, graph classification, $\cdots$).

C.4.2.1 Node classification

In the context of node classification, the last layer of the GNN must output an embedding of dimension p for each node where p is the number of classes (i.e $H^(l) = Z = \mathbb{R}^{|V|\times p}$). For the last layer the activation function is a softmax activation function defined as $softmax(Z_{i,j}) = \frac{\exp(Z_{i,j})}{\sum_{j=1}^{p} \exp(Z_{i,j})}$.

The softmax is applied row-wise and outputs a vector similar to probability distribution over the classes. Then the loss function is the traditional cross entropy.

$$l = -\sum_{(u_i)\in\mathcal{D}}\sum_{j=1}^{p} t_{i,j}\log(Z_{i,j})$$

Where $t_i \in \{0,1\}^p$ is a one-hot vector of the ground-truth class label for node u_i and $\mathcal{D}$ is the data set composed of nodes.

C.4.2.2 Graph classification

In the context of graph classification, a global average pooling layer must be added to gather all the node embeddings of a given graph. This layer has no parameter. It takes as an input $Z = \mathbb{R}^{|V|\times p}$ and output a vector $Z' = \mathbb{R}^{1\times p}$. This layer is performing the average of each component $Z_{.,j}$ over the number of nodes in the graphs. The average pooling layer is defined as $Z'_{1,j} = \frac{1}{|V|}\sum_{i=1}^{|V|} Z'_{i,j} \quad \forall j \in 1,\cdots,p$. This vector Z' can be fed to a MLP for classification. $\hat{t}$ is the prediction made by this extended version of the GNN. Consequently, the cross-entropy loss function can be used.

$$l = -\sum_{(G_i)\in\mathcal{D}}\sum_{j=1}^{\#classes} t_{i,j}\log(\hat{t}_{i,j})$$

Where $\#classes$ is the number of classes. $\mathcal{D}$ is the data set composed of graphs. $t_i \in \{0,1\}^{\#classes}$ is a one-hot vector of the ground-truth class label for graph G_i.

C.4.3 Semi-Supervised

The semi-Supervised node classification problem is the problem of classifying nodes in a graph, where labels are only available for a small subset of nodes. This problem can be framed as semi-supervised learning, where label information is smoothed over the graph via some form of explicit graph-based regularization [Pang and Cheung, 2016], e.g. by using a graph adjacency or graph Laplacian regularization term in the loss function. By using a graph Laplacian/adjacency regularization term, the assumption is that connected nodes in the graph are likely to share the same label. However, this assumption is only true if edges encode the information of node similarity. This is true for instance for a neighborhood graph. The loss function is then closed to the one for fully supervised node classification problem (mentioned earlier) but an adjacency regularization term is added.

$$l = l_0 + \lambda\, l_{reg}$$

$$l_0 = -\sum_{(u_i)\in\mathcal{D}}\sum_{j=1}^{p} t_{i,j}\log(Z_{i,j})$$

$$l_{reg} = \sum_{(u_i,u_j)\in\mathcal{D}} (Z_i^T Z_j - A_{i,j})^2 = vec(Z)^T A\, vec(Z)$$

To accomplish a graph Laplacian regularization, A must be replaced by $\Delta = D - A$ that denotes the unnormalized graph Laplacian of an undirected.

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Figure C.3: Statistics on citation networks (From a T. Kipf's talk http://deeploria.gforge.inria.fr/thomasTalk.pdf).

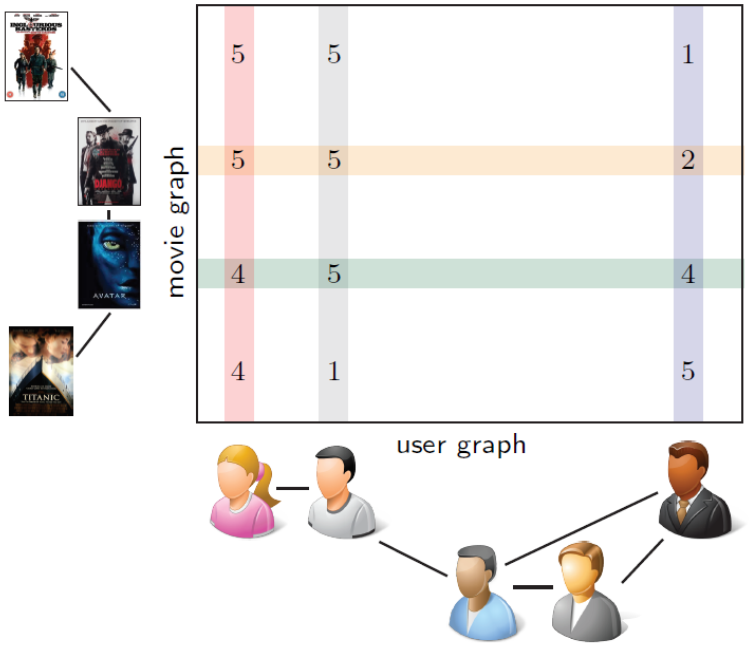


Figure C.4: Structured matrix completion (From CVPR 2017 tutorial http://geometricdeeplearning.com/).

C.4.4 Applications

C.4.4.1 Node classification:

Node classification on citation networks. The input is a citation network where nodes are papers, edges are citation links and optionally bag-of-words features on nodes. The target for each node is a paper category (e.g. stat.ML, cs.LG, ...). Datasets statistics are pictured in Figure C.3.

C.4.4.2 Graph regularization:

Matrix completion is the task of filling in the missing entries of a partially observed matrix. A wide range of datasets are naturally organized in matrix form. One example is the movie-ratings matrix, as appears in the Netflix problem: Given a ratings matrix in which each entry (i,j) represents the rating of movie j by user i. When users and movies are organized as graphs (Pictorially Figure C.4) then graphs can be used to regularized the matrix completion problem.

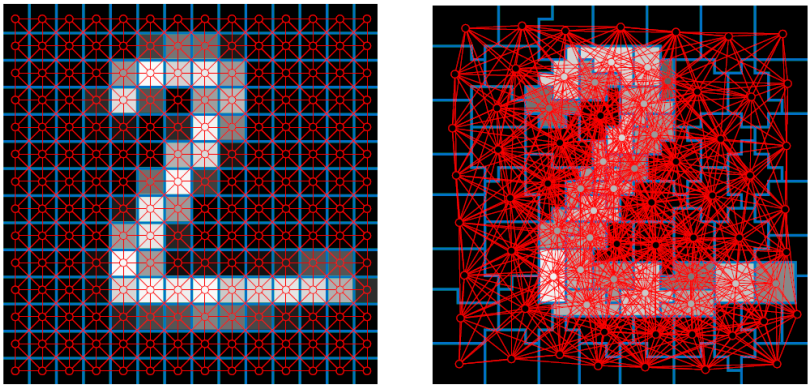


Figure C.5: MNIST digits classiffication (Regular grid, Superpixels) (From CVPR 2017 tutorial http://geometricdeeplearning.com/).

C.4.4.3 Graph classification:

An image is represented as a graph: based on raw pixels (a regular grid and all images have the same graph) or based on superpixels (irregular graph) (see Figure C.5).

C.4.4.4 Datasets and results:

Two main sources of graphs can be found in the literature : bioinformatics and social networks (see Table C.1 for summary statistics of these datasets)

Social networks datasets. IMDB-BINARY and IMDB-MULTI are movie collaboration datasets. Each graph corresponds to an ego-network for each actor/actress, where nodes correspond to actors/actresses and an edge is drawn betwen two actors/actresses if they appear in the same movie. Each graph is derived from a pre-specified genre of movies, and the task is to classify the genre graph it is derived from. REDDIT-BINARY and REDDIT-MULTI5K are balanced datasets where each graph corresponds to an online discussion thread and nodes correspond to users. An edge was drawn between two nodes if at least one of them responded to another's comment. The task is to classify each graph to a community or a subreddit it belongs to. COLLAB is a scientific collaboration dataset, derived from 3 public collaboration datasets, namely, High Energy Physics, Condensed Matter Physics and Astro Physics. Each graph corresponds to an ego-network of different researchers from each field. The task is to classify each graph to a field the corresponding researcher belongs to.

Bioinformatics datasets. MUTAG is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds with 7 discrete labels. PROTEINS is a dataset where nodes are secondary structure elements (SSEs) and there is an edge between two nodes if they are neighbors in the amino-acid sequence or in 3D space. It has 3 discrete labels, representing helix, sheet or turn. PTC is a dataset of 344 chemical compounds that reports the carcinogenicity for male and female rats and it has 19 discrete labels. NCI1 is a dataset made publicly available by the National Cancer Institute (NCI) and is a subset of balanced datasets of chemical compounds screened for ability to suppress or inhibit the growth of a panel of human tumor cell lines, having 37 discrete labels.

The state-of-the-art baselines for graph classification: (1) the WL subtree kernel [Shervashidze et al., 2011] with C-SVM was used as a classifier. (2) state-of-the-art deep learning architectures,

Dataset	Size	Classes	Avg.nodes	Labels
MUTAG	188	2	17.9	7
PTC	344	2	25.5	19
ENZYMES	600	6	32.6	3
PROTEINS	1113	2	39.1	3
NCI1	4110	2	29.8	37
NCI109	4127	2	29.6	38
COLLAB	5000	3	74.49	-
IMDB-BINARY	1000	2	19.77	-
IMDB-MULTI	1500	3	13	-
REDDIT-BINARY	2000	2	429.61	-
REDDIT-MULTI-5K	5000	2	508.5	-
REDDIT-MULTI-12K	11929	11	391.4	-

Table C.1: Properties of the Bioinformatics and Social network datasets used in graph/node classification experiments.

Datasets	IMDB-B	IMDB-M	RDT-B	RDT-M5K	COLLAB	MUTAG	PROTEINS	PTC	NCI1
# graphs	1000	1500	2000	5000	5000	188	1113	344	4110
# classes	2	3	2	5	3	2	2	2	2
Avg # nodes	19.8	13.0	429.6	508.5	74.5	17.9	39.1	25.5	29.8
Baselines									
WL subtree	73.8 ± 3.9	50.9 ± 3.8	81.0 ± 3.1	52.5 ± 2.1	78.9 ± 1.9	90.4 ± 5.7	75.0 ± 3.1	59.9 ± 4.3	86.0 ± 1.8
GNN variants									
MEAN–1-LAYER (GCN)	74.0 ± 3.4	51.9 ± 3.8	50.0 ± 0.0	20.0 ± 0.0	79.0 ± 1.8	85.6 ± 5.8	76.0 ± 3.2	64.2 ± 4.3	80.2 ± 2.0
MAX–1-LAYER (GraphSAGE)	72.3 ± 5.3	50.9 ± 2.2	–	–	–	85.1 ± 7.6	75.9 ± 3.2	63.9 ± 7.7	77.7 ± 1.5

Table C.2: Test set classification accuracies (%).

i.e., (GCN) [Kipf and Welling, 2016], (GraphSage) [Hamilton et al., 2017].

Table C.2 compares test accuracies of the methods.

C.5 Advanced GNN

C.5.1 Gated GNN

GNNs and GraphSAGE generally are only 2-3 layers deep. Increasing the number of layers may lead to an overfitting phenomenon and the vanishing/exploding gradients during backpropagation. Is it possible to go deeper? A proposal to solve this problem is described in [Li et al., 2015]. Going deeper is possible if the number of parameters is reduced. A solution is to share parameters across layers. It means that a single neural network is used, the same for each layer. To take into account that the layer l impacts the layer $l+1$, a Recurrent Neural Network (RNN) is used where the notion of time is replaced by the concept of layer. The global idea is that nodes aggregate "messages" from their neighbors using a recurrent neural network. This idea is depicted in Figure C.6. The new layer $l+1$ is computed by the taking "message" from neighbors at step l as well as the output of the layer l.

$$H^{(l+1)} = RNN(H^{(l)}, AGG^{(l)})$$

This architecture can handle models with > 20 layers and it allows for complex information about global graph structure to be propagated to all nodes.

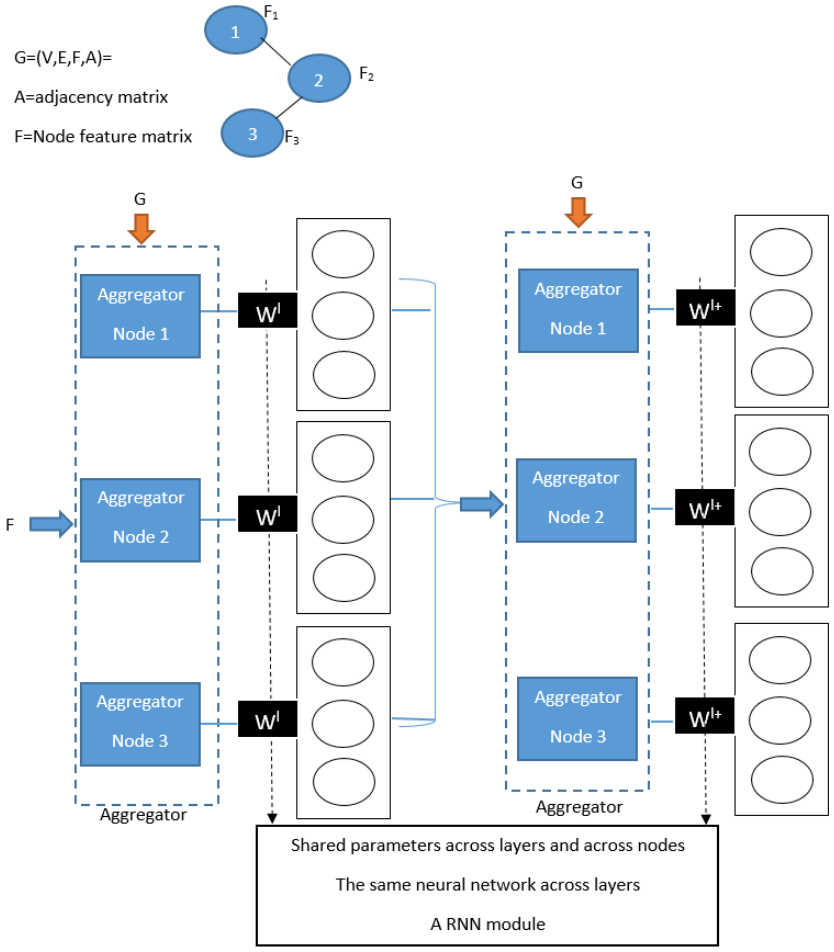


Figure C.6: A general overview of a Gated GNN.

C.5.2 Graph pooling

Each layer H^l outputs a graph and a node embedding. So far the graph is the same as the input graph. To extract information at different scales two schemes can be adopted:

- Adjacency operators at different scales (A^J) as in [Nowak et al., 2017].
- Graph coarsening for graph pooling as in [Monti et al., 2016].

The graph pooling goals are: a) pool similar local features (max pooling or average pooling) and b) series of pooling layers create invariance to global geometric deformations. The main challenge is to design a multi-scale coarsening algorithm that preserves non-linear graph structures. Graph coarsening decomposes G into smaller meaningful clusters. This problem is combinatorial and is $\mathcal{NP}$-hard.

In [Monti et al., 2016], a graph pooling layer is added. The pooling layer takes as input $[G, H^l]$ and outputs $[G', H^{l+1}]$ where $|V'| < |V|$ and $H^{l+1} \in \mathbb{R}^{|V'| \times m_l}$. In the graph pooling layer, some graph clustering algorithm have be favored such as [Dhillon et al., 2007, Blondel et al., 2008]. Graph pooling layers are inspired from pooling layers that appear in classical Convolutional Neural Network (CNN).

C.5.3 Differentiation and training

Training a GNN is similar to standard MLP or CNN. The main idea is to minimize the loss $l()$ (also called cost function) according to the parameters. The minimum of the cost function is where its derivative is equal to 0.

$$\frac{\partial l(W,G)}{\partial W} = 0$$

C.5.3.1 Derivative (or gradient) computation

The loss is a composition of functions so its derivative is a composition of derivatives. This principle is called the chaine rule or back propagation.

$$\frac{\partial l(W,G)}{\partial W} = \frac{\partial l(W,G)}{\partial H^{(L)}(W)} \times \frac{\partial H^{(L)}(W)}{\partial H^{(L-1)}(W)} \times \frac{\partial \partial H^{(L-1)}(W)}{\partial H^{(L-2)}(W)} \cdots$$

If each function that composed the neural network is differentiable then the derivative can be computed. Issues occur for functions that are not continuous anywhere. This is the case of ReLU (Rectified Linear Units) $ReLU(x) = \max(0, x)$. The derivative is then :

$$ReLU'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{C.1}$$

Now what about $x = 0$? Technically this is undefined. When $x = 0$, there are many possible lines (slopes) we could fit through it. So what to do here?

Basically, it is commonly accepted to impose a slope when x=0. A common choice is when x=0, the derivative will be 0. It could be some other value, but most implementations use this (this has a nice property that it encourages many values to be 0 i.e., sparsity in the feature map). By doing so the $ReLU$ function is modified but apparently it does not impact the training algorithms. The same phenomenon appears for the max pooling function. This function selects one node of G to be part of G' in the graph pooling layer. Nodes from G that are not in G' will have their derivative set to 0.

C.5.3.2 Gradient descent

The question is finally to find the parameters W_{min} that leads to $\frac{\partial l(W_{min},G)}{\partial W_{min}} = 0$. To achieve this goal, the gradient descent algorithm is often selected. The algorithm is depicted in Algorithm 4. The main feature of this algorithm is to update W iteratively in the opposite direction of the gradient. The algorithm as one parameter called α. This scalar controls the speed of the descent. High values of α can lead to non-convergence. Based on this paradigm more complex methods has arisen [Kingma and Ba, 2014, Marceau-Caron and Ollivier, 2016].

Algorithm 4 Graph descent algorithm

Input: $\#iter$ is the maximum number of iterations.
α is the learning rate controlling the descent step.
Output: W_{min}. Learned W. Weight matrices

1: $W \leftarrow random$ and $i \leftarrow 0$
2: **while** i< $\#iter$ **do**
3: $W(i+1) \leftarrow W(i) - \alpha \frac{\partial l}{\partial W}$
4: i ← i +1
5: **end while**

C.5.4 A word on scalability and time complexity?

Neural networks in general are successful because they can take advantage of the computational power provided by CPU and GPU. In this direction, operations within a layer should be vectorized. Vectorization is the process of converting an algorithm from operating on a single value at a time to operating on a set of values at one time. Vector, matrix or tensor operations are easily vectorized. In some final applications, the adjacency matrix can be sparse and there is special representations and libraries for such matrices. For instance, the product AF can be efficiently implemented as a product of a sparse matrix with a dense matrix .

Secondly, computational complexity should be low. In [Kipf and Welling, 2016], the model is trainable in $\mathcal{O}(|E|)$ time. Operators **I**, A, D and U decribed in [Nowak et al., 2017] have the same complexity while the operator A^{2^j} is much more greedy.

C.5.5 Spatial vs spectral convolution on graphs

There is no consensus on the convolution (operator $*$) definition on non-euclidean signals. Two different types of definitions arise from the literature [Bronstein et al., 2016].

C.5.5.1 Relation between spatial and spectral convolution

Given two functions, f and g : $[-\pi, \pi] \rightarrow \mathbb{R}$ their convolution is a function

$$(f * g)(x) = \int_{-\pi}^{\pi} f(x')g(x - x')dx'$$

Fourier transform diagonalizes the convolution operator ⇒ Convolution can be computed in the Fourier domain as:

$$\widetilde{(f * g)} = \tilde{f}.\tilde{g}$$

Where $\tilde{.}$ stands for the function being in the Fourier domain.

In matrix-vector notation, with the $n \times n$ Fourier matrix $\phi = [\phi_1, \cdots, \phi_n]$:

$$\tilde{f} = \phi^T f \text{ and } f = \phi \tilde{f}$$

Convolution of two vectors: $f = (f_1, \cdots, f_n)^T$ and $g = (g_1, \cdots, g_n)^T$

$$g * f = \begin{pmatrix} g_1 & g_2 & \cdots & g_n \\ g_n & g_1 & \cdots & g_{n-1} \\ \vdots & & \ddots & \vdots \\ g_2 & g_3 & \cdots & g_1 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$$

$$g * f = \phi \begin{pmatrix} \tilde{g_1} & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \tilde{g_n} \end{pmatrix} \phi^T f$$

$$g * f = \phi \text{ diag}(\tilde{g_1}, \cdots, \tilde{g_n}) \phi^T f$$

C.5.5.2 Spectral convolution for graphs

Convolution will be noted as $G_I * f$ with G_I the input graph and f the filter in its abstract form.

The first one is based on spectral graph theory, which is the equivalent of Fourier analysis for graphs. The idea is to take advantage of the Convolution Theorem to apply convolution on a given graph. The convolution theorem states that convolution in the spatial domain is equivalent to product in the frequency domain. It is then possible to multiply the two signals in the frequency domain to obtain the convolved signal. The structure that allows to work on the frequency domain of a graph is its Laplacian:

$$L = D - A$$

with D and A respectively being the degree and adjacency matrices.

The eigenvectors of the graph Laplacian act as the Fourier basis of the graph. Convolving over a graph from its frequency domain is as follows:

$$G_I * f = \Phi_I diag(\tilde{f}) \Phi_I^\intercal M_I$$

where $\tilde{f}$ is the filter f in the spectral domain, M_I is the vector of edge attributes in graph G_I and Φ_I are the Laplacian eigenvectors for graph G_I [Bronstein et al., 2016].

This method is used in [Bruna et al., 2013]. However, getting access to the eigenbasis of the graph laplacian is time-consuming as it implies matrix inversion.

A way to avoid eigenanalysis is to consider our filter f as a polynomial of the Laplacian $f_\theta(L_I)$ (L_I is the laplacian for graph G_I) [Bronstein et al., 2016, Defferrard et al., 2016]:

$$G_I * f = f_\theta(L_I) M_I$$

$$f_\theta(L) = \sum_{k=0}^{r} \theta_k . L^k$$

[Kipf and Welling, 2016, Defferrard et al., 2016, Scarselli et al., 2009] use this method, introducing different types of polynomials.

The major drawback of spectral approaches is sensitivity to domain changes: Similar graphs with slightly different topologies will respond very differently to a given filter. In other words, a single signal defined on anisomorph graphs will give different responses. This is due to the fact that constructing the Fourier basis depends on the graph structure. The second type of graph convolution definition relies solely on spatial domain. The common point of these approaches is to consider the problem in a riemannian perspective: non-euclidean data can be seen as manifolds. This allows to reduce the local structure of graphs down to a compact euclidean space on which filters can be easily built: a patch operator $\mathbb{D}$ can be defined. The patch operator transforms the local space into a P-sized vector with respect to P weighting functions $(w_1, \ldots, w_P)$. $w_k(i)$ is the vector of weights for each node in the graph in the neighbourhood of i. The patch value at node i is defined as follows:

$$(\mathbb{D}G_I)_i = [w_k(i)M_I]_{1 \leq k \leq P}$$

These weights are defined depending on the locations of the different neighbours in a given local space.

$$w_k(i) = [e_k(\zeta_{ii'})]_{1 \leq i' \leq N}$$

where N is the number of nodes in the graph, $\zeta_{ii'}$ is the label for edge (i, i') (or the location of i' in the local space of i) and $e_k(\zeta_{ii'})$ is the weight for node i' in the neighbourhood of i.

Once our patch operator is applied, it is possible to define our convolution filter f as a vector $\mathbf{f} \in \mathbb{R}^P$. The result of the convolution at node i is as follows:

$$(G_I * f)_i = (\mathcal{D}G_I)_i.\mathbf{f}$$

Several approaches fit in this spatial framework [Boscaini et al., 2016, Monti et al., 2016]. The way they differ is how they define the weighting functions e_k.

[Monti et al., 2016] proposes to use parameterized gaussian kernels as weight functions:

$$e_k(\zeta_{ii'}) = \exp -\frac{1}{2}(\zeta_{ii'} - \mu_k)\Sigma_k^{-1}(\zeta_{ii'} - \mu_k)$$

Where μ_k and Σ_k are the gaussian kernel parameters for the k-th weighting function. These parameters can be learned during the training of the neural network as it is included in the loss function and it is differentiable.

Nevertheless, most of the aforementioned approaches don't take advantage fully of the graph topology. The graph structure is locally embedded into a vector space (i.e. the tangent space at a given point of a riemannian manifold). In a graph theory perspective, this means the notion of neighbourhood is limited to 1-hop graph neighbourhood which is bound to be a star structure.

C.6 Summary on GNN

The key idea is to generate node embeddings based on local neighborhoods.

- Graph convolutional networks
  - Average neighborhood information and stack neural networks.
- GraphSAGE

  - Generalized neighborhood aggregation.

- Gated Graph Neural Networks
  - Neighborhood aggregation + RNNs

- Model wish list :
  - Set W^l of trainable parameters
  - Trainable linear in time in function of $|E|$ or $|V|$.
  - Applicable even if the input graph changes

Appendix D

Kernels and kernel machines for graphs

The similarity between graphs are either based on a vector representation or defined directly in the space of graphs. Another option is to project the graphs in a kernel space. The kernels provide a mathematical framework for defining a measure of similarity between objects corresponding to a scalar product in a vector space that is not necessary known explicitly. Therefore, the kernels make it possible to overcome the limits induced by fixed size vectors while allowing the use of statistical learning methods. However, the definition of a kernel is not trivial and must respect a set of conditions to define a scalar product.

D.1 Kernel theory

Let define a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ between two objects x and x' corresponds to a scalar product between two projections $\phi(x)$ and $\phi(x')$ in a Hilbert space $\mathcal{H}$.

$$\forall (x, x') \in \mathcal{X} \times \mathcal{X}, \quad k(x, x') =< \phi(x), \phi(x') >$$

In order to define a valid kernel, it is not necessary to explicitly define the projection function $\phi : \mathcal{X} \to \mathcal{H}$. However, the kernel k must verify certain properties:

Definition 27. *(Positive-definite kernel)*
A positive-definite kernel on $\mathcal{X} \times \mathcal{X}$ is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$:

$$k(x, x') = k(x', x)$$

and semi-definite positive:

$$\{x_1, \cdots, x_M\} \in \mathcal{X}^M, c \in \mathbb{R}^M, \sum_{i=1}^{M} \sum_{j=1}^{M} c_i k(x_i, x_j) c_j \geq 0$$

Definition 28. *(Gram matrix)*
A Gram matrix $K \in \mathbb{R}^{M \times M}$ associated to a kernel k on a finite set $X = \{x_1, \cdots, x_M\}$

$$K_{i,j} = k(x_i, x_j), (i, j) \in \{1, ..., M\}^2$$

If k is a positive-definite kernel then the Gram matrix K is semi-definite positive. The reverse is also true.

D.1.1 Kernel and dissimilarity

In pattern recognition, it is desired to combine a kernel with the k-nearest neighbors algorithm. It is therefore necessary to calculate distances. Kernels can be used to construct dissimilarities.

$$\begin{aligned} d(x,x') &= ||\phi(x) - \phi(x')||_2 \\ &= <\phi(x),\phi(x)> + <\phi(x'),\phi(x')> -2<\phi(x),\phi(x')> \\ &= k(x,x) + k(x',x') - 2k(x,x') \end{aligned}$$

Therefore, the k-nearest neighbors algorithm can be applied in the vector space without having to calculate the projections $\phi(x)$ and $\phi(x')$ but only the value of the kernel $k(x,x')$. This property is called the kernel trick.

D.2 Kernel machines

Machine learning techniques that involve kernels are called Kernel machines. These algorithms include, support vector machine, nearest-neighbor classifier, principal component analysis, Fisher discriminant analysis, k-means clustering, and many more.

The **k-nearest neighbors** algorithm where the distance function is based on a kernel can be called a Kernel machine. One of the significant limitations of such algorithms is that the kernel function $k(x,x')$ must be evaluated for all possible pairs x and x' of training points, which can be computationally infeasible during training and can lead to excessive computation times when making predictions for new data points.

Kernel machines that have sparse solutions, so that predictions for new inputs depend only on the kernel function evaluated at a subset of the training data points. The **support vector machine** (SVM), which became popular in some years ago for solving problems in classification, regression, and novelty detection. An important property of support vector machines is that the determination of the model parameters corresponds to a convex optimization problem, and so any local solution is also a global optimum.

D.2.1 Distance to the decision line

Let us define a linear function $\hat{t} = f(x,w,b) = w^T x + b \in \mathbb{R}$ (see Figure D.1). The distance between a point x and the line represented by the function $f()$ is defined by:

$$d^{\perp}(x, f(.)) = \frac{f(x,w,b)}{||w||_2} \quad \text{(Orthogonal projection)}$$

Let us define the true target $t \in \{-1, 1\}$ to distinguish between two classes. A sample x is correctly classified if $f(x,w,b) = t$ or $t.f(x,w,b) > 0$. The distance of correctly classified sample to the line represented by the function $f()$ is defined by:

$$d(x, f(), t) = \frac{t.f(x,w,b)}{||w||_2}$$

D.2.2 Finding the margin

Among a data set $\mathcal{D} = \{(x_1,t_1), \cdots, (x_M,t_M)\}$, the margin is the smallest distance between the decision line $(f(.))$ and the samples. Finding the closest sample to the function f is then defined by:

$$d_{min} = \min_{i \in [1,\cdots,M]} \frac{t.f(x_i,w,b)}{||w||_2}$$

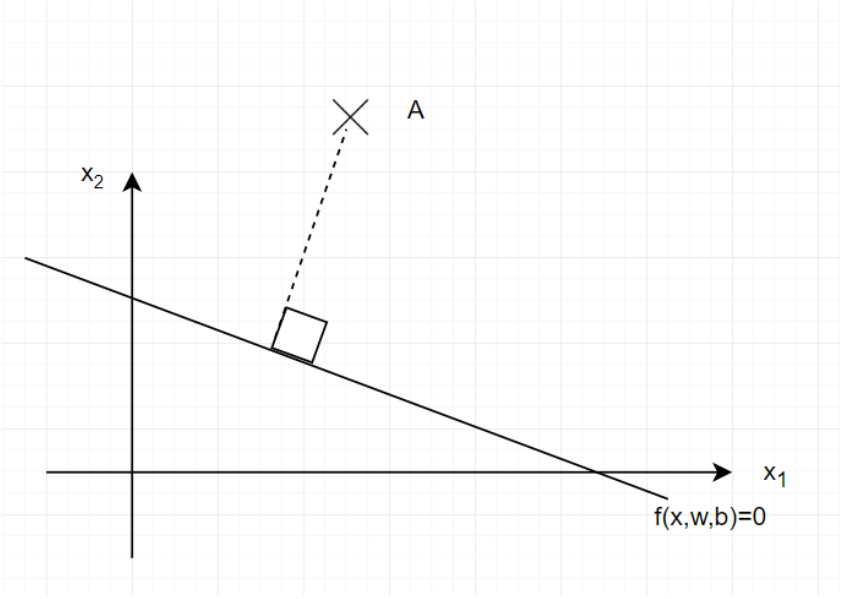


Figure D.1: Orthogonal projection of the point A on the line.

D.2.3 Maximizing the margin

A key idea of SVM is to find the margin which is the largest. As the data cannot be modified, parameters w and b must be tuned to maximize d_{min}:

$$arg \max_{w,b} \quad d_{min}(w,b) \tag{D.1}$$

$$arg \max_{w,b} \quad \left[\frac{1}{||w||_2}(\min_{i\in[1,\cdots,M]} w^T x_i + b)\right] \tag{D.2}$$

Equation D.2 is the heart of the SVM problem finding the parameters that maximize the margin. However, this formulation is not easy to optimize.

D.2.4 Rescaling the parameters

To make the problem easier to solve, a rescaling of the data must be performed. $w = cst.w$ and $b = cst.b$ where cst is a constant then the rescaling does not change the distance $d(x, f(), t)$. If the closest sample $x_{min} = arg\ \min_{i\in[1,\cdots,M]} \frac{t.f(x_i,w,b)}{||w||_2}$, we can use this freedom to set $t_{min}(w^T x_{min} + b) = 1$. And all the samples satisfy:

$$t_i(w^T x_i + b) = 1 \geq 1, \forall i \in [1, \cdots, M]$$

D.2.5 Canonical formulation

Maximizing $\frac{1}{||w||_2}$ is equivalent to minimize $\frac{1}{2}||w||_2^2$

$$arg \min_{w,b} \quad \frac{1}{2}||w||_2^2 \tag{D.3}$$

$$Subject\ to \quad t_i(w^T x_i + b) \geq 1 \quad \forall i \in [1, \cdots, M] \tag{D.4}$$

The determination of the model parameters corresponds to a convex optimization problem. This is an example of a quadratic programming problem in which we are trying to minimize a quadratic function subject to a set of linear inequality constraints.

D.2.6 Lagrangian formulation

In order to solve this constrained optimization problem, we introduce Lagrange multipliers an $\alpha_i \geq 0$, with one multiplier a_i for each of the constraints ($a = (a_1, \cdots, a_M)$).

$$L(a, w, b) = \frac{1}{2}||w||_2^2 - \sum_{i=1}^{M} a_i \left[t_i(w^T x_i + b) - 1\right]$$

Note the minus sign in front of the Lagrange multiplier term, because we are minimizing with respect to w and b, and maximizing with respect to a.

D.2.7 Dual formulation and kernel formulation

From the langrangian function L(.), we want to find its minimum according to parameters w and b so we need to find where its derivative equal 0.

$$\frac{\partial L}{\partial w} = 0 = w - \sum_{i=1}^{M} a_i t_i x_i \implies w = \sum_{i=1}^{M} \alpha_i t_i x_i$$

$$\frac{\partial L}{\partial b} = 0 = \sum_{i=1}^{M} a_i t_i$$

By substituting $w = \sum_{i=1}^{M} a_i t_i x_i$ in the function $L(.)$, we obtain the dual formulation:

$$\tilde{L}(a) = \sum_{i=1}^{M} a_i - \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} a_i a_j t_i t_j x_i x_j$$

By replacing the dot product $x_i x_j$ by the kernel $k = (x_i, x_j)$, we obtain the SVM kernel machine:

$$arg \max_a \tilde{L}(a) = \sum_{i=1}^{M} a_i - \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} a_i a_j t_i t_j k(x_i, x_j) \tag{D.5}$$

$$Subject\ to \tag{D.6}$$

$$a_i \geq 0, \forall i \in [1, \cdots, M] \tag{D.7}$$

$$\sum_{i=1}^{M} a_i t_i = 0, \forall i \in [1, \cdots, M] \tag{D.8}$$

D.2.8 Solving SVM

Solutions for this optimization problem either: (i) reduce it to an equivalent polynomial-size reformulation (for certain decomposable loss functions), and use methods like SMO (sequential minimal optimization) [Taskar et al., 2003] or general-purpose solvers; or (ii) work with the original problem by considering a subset of constraints, and employing cutting plane [Tsochantaridis et al., 2005] or stochastic subgradient methods. The solution to a quadratic programming problem in d variables in general has computational complexity that is $O(d^3)$. In going to the dual formulation, the original optimization problem, which involved minimizing over d variables, into the dual problem, which has M variables. For a fixed set of basis functions whose number d is smaller than the number M of data points, the move to the dual problem appears disadvantageous. However, it allows the model to be reformulated using kernels, and so the maximum margin classifier can be applied efficiently to feature spaces whose dimensionality exceeds the number of data points.

D.2.9 Classification

In order to classify new data points using the trained model, the sign of $f(x, w, b)$ is evaluated, defined by $\hat{t} = w^T x + b$, if the canonical formulation is solved (Equation D.4). Another possibility can be expressed in terms of the parameters $\{a_n\}$ and the kernel function if dual formulation was solved (Equation D.8) :

$$\hat{t} = \sum_{i=1}^{M} a_i t_i k(x, x_i) + b$$

Any data point for which $a = 0$ will not appear in the sum in and hence plays no role in making predictions for new data points. The remaining data points are called *support vectors*. This property is central to the practical applicability of support vector machines. Once the model is trained, a significant proportion of the data points can be discarded and only the support vectors retained.

$$\hat{t} = \sum_{i \in S} a_i t_i k(x, x_i) + b$$

Where S denotes the set of indices of the support vectors.

D.3 Graph mathing-based Kernels

On the basis that a similarity measure can be defined as a decreasing function of a dissimilarity measure, some kernels are defined from a distance between graphs. The graph edit distance between graphs, measures the dissimilarity between graphs: a distance high indicates a low similarity between the two graphs while a low distance indicates a strong similarity.

Kernel functions that are derived from graph edit distance. Based on the assumption that graph edit distance is well suited for difficult graph matching problems, kernel functions are proposed that are sufficiently flexible for unconstrained graph representations. Regarding kernel functions as similarity measures, we obtain embeddings of the space of graphs into vector spaces, where the similarity of vectors is defined according to the edit distance of the original graphs.

D.3.1 Trivial GED kernels

Trivial kernels [Neuhaus and Bunke., 2007] can be defined:

$$\begin{aligned} k1(G_1, G_2) &= -GED(G_1, G_2) \\ k2(G_1, G_2) &= -GED(G_1, G_2)^2 \\ k3(G_1, G_2) &= -\tanh(-GED(G_1, G_2)) \\ k4(G_1, G_2) &= \exp\left(-GED(G_1, G_2)\right) \end{aligned}$$

However, the edit distance between two graphs does not define a metric in a Euclidean space, so there is no guarantee that the associated distance matrices will be negative-definite. The Gram matrices calculated by the GED kernels are therefore not semi-definite positive [Neuhaus and Bunke., 2007]. Therefore, the use of the kernel machines is limited since the function to be minimized is no longer convex and therefore no guarantee of convergence towards the global minimum is ensured. However, non-compliance with the semi-defined positivity of the kernel does not completely exclude the use of kernel machines.

D.3.2 Kernels based on GED embedding [Riesen and Bunke, 2010b]

The explicit graph embedding defined by $\phi(G) = [GED(G,G_1), \cdots, GED(G,G_M)]$ can endow a simple $k5 = (G_1, G_2) =< \phi(G_1), \phi(G_2) >$. Considering this embedding function, the kernel between two graphs is simply defined as the dot product between their embeddings. This method therefore uses an explicit vector representation to define a kernel based on editing distance. Although this method uses a dissimilarity measure widely used in the field of comparison graphs, the choice of reference graphs strongly influences the quality of the embedding.

D.3.3 Kernel from Maximum-Similarity Edit Path

In [Neuhaus and Bunke., 2007], a method is proposed to re-formulate graph edit distance as a graph similarity measure. The idea is to turn the minimum-cost edit path condition into a maximum-similarity edit path criterion.

$$k(G_1, G_2) = \max_{\lambda \in \Gamma'(G_1, G_2)} \Pi_{(u \to v) \in \lambda} k(\mu_1(u), \mu_2(v))$$

Where $\Gamma'(G_1, G_2)$ is obtained from by the set of all edit path and then by removing all deletions and insertions of nodes and edges. Hence, the kernel function only considers the substitution of nodes and edges. The computation of this kernel function can be carried out by means of a modified edit distance algorithm.

D.3.4 Convolution kernels and local matching kernels [Neuhaus and Bunke., 2007]

The idea is to decompose complex objects into smaller parts, for which a similarity function can more easily be defined or more efficiently be computed. Using a convolution operation, these similarities are then turned into a kernel function on the composite objects. Let us introduce the decomposition $R(G) = \{g_1, \cdots, g_d\}$. For a simple example, assume that the set of all decompositions of a graph $R(G) = V$. This means that each of its nodes is a valid decomposition of G. The corresponding convolution kernel is then:

$$k(G_1, G_2) = \sum_{u \in R(G_1)} \sum_{v \in R(G_2)} k(u, v) = \sum_{\substack{u \in R(G_1) \\ v \in R(G_2)}} k(u, v)$$

It simply returns the sum of similarity between pairs of nodes.

Let us define $R(G) = \{(s, (u_i, n_i, e^1_{u_i}, \cdots, e^{n_i}_{u_i}))|\forall i \in 1, \cdots s_{max}\}$. The first component s specifies the number of nodes $u_1, \cdots, u_s$ present in the decomposition, up to a maximum number of nodes s_{max}. The remaining components of type $(u_i, n_i, e^1_{u_i}, \cdots, e^{n_i}_{u_i})$ represent a node u_i, together with n_i of the $|E_{u_i}|$ edges originating in u_i. Clearly, components $(u_i, n_i, e^1_{u_i}, \cdots, e^{n_i}_{u_i})$ can be regarded as substructures of G each consisting of a single node and some, possibly not all, of its adjacent edges. These local substructures of graphs, or partial subgraphs, can then be used for graph matching. The local matching convolution kernel is defined by:

$$k(G, G') = \sum_{\substack{(s,(u_i,n_i,e^1_{u_i},\cdots,e^{n_i}_{u_i})) \in R(G) \\ (s',(u'_i,n'_i,e'^1_{u'_i},\cdots,e'^{n'_i}_{u'_i})) \in R(G')}} k_\delta(s, s') \Pi_{i=1}^{s} k(u_i, u'_i) k_\delta(n_i, n'_i) \Pi_{p=1}^{n_i} k(e^p_{u_i}, e'^{p'}_{u'_i})$$

It is clear that for two decompositions with a different number of nodes $s \neq s'$, the resulting product will be zero, due to the Dirac kernel, $k_\delta(s, s') = 0$. Similarly, if a node u_i in the decomposition contains a different number of edges than the corresponding node in the other graph u'_i,

$n_i \neq n'_i$, the resulting similarity will be zero as well, due to $k_\delta(n_i, n'_i)$. Hence, only the similarity of decompositions that are consistent in terms of the number of nodes and edges are taken into account, that is, contribute a non-zero similarity value to the convolution kernel.

D.3.5 Random Walk Edit Kernel [Neuhaus and Bunke., 2007]

Another class of graph kernels is based on the evaluation of random walks in graphs. These kernels measure the similarity of two graphs by the number of (possibly infinite) random walks in both graphs that have all or some labels in common.

The basic idea of random walk kernels is to define the similarity of graphs by comparing random walks in two graphs. For instance, one of the most elegant random walk kernels computes the number of matching random walks in two graphs. A key observation is that this computation can efficiently be realized by means of the direct product of two graphs, without having to explicitly enumerate random walks in graphs. This enables to consider random walks of arbitrary length. Random walk kernels are undoubtedly very efficient, but on noisy data their accuracy is often unsatisfactory. For this reason, an extension to a standard random walk kernel is proposed in this section to make the kernel more robust and therefore applicable to noisy graph data as well. The idea is to integrate information from the global matching of graphs into the otherwise locally defined random walk kernel. To this end, we first compute the edit distance of graphs and use the optimal edit path to define the adjacency matrix of the direct product in an extended way to enhance the robustness of the random walk kernel.

Definition 29. *(Direct graph product)*
$V_X = \{(i,k) | \mu(i) = \mu(k) \, \forall i \in V_1, \, k \in V_2\}$
$E_X = \{((i,k),(j,l)) | \zeta((i,k)) = \zeta((j,l)) \, \forall (i,j) \in E_1, \, (k,l) \in E_2\}$

Definition 29 is restricted to evaluating whether two discrete attributes are identical or not. An extension of the direct graph product is welcome to consider rich attributed graphs.

Definition 30. *(Modified direct graph product)*
$V_X = \{(i,k) | \, \forall i \in V_1, \, k \in V_2\}$
$E_X = \{((i,k),(j,l)) | \, \forall (i,j) \in E_1, \, (k,l) \in E_2\}$

The adjacency matrix A_X of this modified direct product graph can then be defined by :

$$A_{x(i,k),(j,l)} = \begin{cases} k_{edge}((i,k),(j,l)) & \text{if } ((i,k),(j,l)) \in E_X \\ 0 & \text{Otherwise} \end{cases}$$

Where the kernel function k_{edge} measuring the similarity of pairs of nodes and edges based on edit operations. The random walk kernel enhanced by edit distance is then defined according to the standard random walk kernel function. Given a decay factor $0 < \Lambda < 1$:

$$k(G_1, G_2) = \sum_{i=1}^{|V_x|} \sum_{j=1}^{|V_x|} \left[\sum_{n=0}^{\infty} \Lambda^n A_x^n \right]_{ij}$$

The edit distance enhanced random walk kernel differs from the random walk kernel for discretely labeled graphs and the modified random walk kernel for continuously labeled graphs only in the definition of the adjacency matrix A_X of the direct product.

D.3.6 Diffusion kernel

In [Neuhaus and Bunke., 2007], the edit distance between graphs ($\mathcal{D} = \{(G_1, t_1), \cdots, (G_M, t_M)\}$) is turned into a matrix of non-negative similarities:

$$B_{i,j} = \max_{1 \leq s,t \leq M} (GED(G_s, G_t)) - GED(G_i, G_j) \quad \forall i, j \quad 1 \leq i, \quad j \leq M$$

where $B_{i,j}$ denotes the similarity of graphs G_i and G_j .

Given a decay factor $0 < \Lambda < 1$, the exponential diffusion kernel is defined by:

$$K = \sum_{k=0}^{\infty} \frac{1}{k!} \Lambda^k B^k = exp(\Lambda B)$$

K is a Gram matrix.

Appendix E

Publications

E.1 Liste des publications

E.1.1 Thèse

- Raveaux [2010] Romain Raveaux. *Graph Mining and Graph Classification : application to cadastral map analysis*. Theses, Université de La Rochelle, November 2010. URL https://tel.archives-ouvertes.fr/tel-00567218

E.1.2 Chapitre de livres

- Raveaux et al. [2013b] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. La vectorisation automatisée des plans Vasserot, dans Paris de parcelles en pixels. In *Analyse géomatique de l'espace parisien médiéval et moderne*, pages 54–65. Presse Universitaire de Vincennes, April 2013b. URL https://hal.archives-ouvertes.fr/hal-00936555

E.1.3 Revues Internationales

- Abu-Aisheh et al. [2018, in press] Zeina Abu-Aisheh, Romain Raveaux, and Jean-Yves Ramel. Efficient k-nearest neighbors search in graph space. *Pattern Recognition Letters*, 2018, in press. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2018.05.001. URL http://www.sciencedirect.com/science/article/pii/S0167865518301673

- Martineau et al. [2018, in press] Maxime Martineau, Romain Raveaux, Donatello Conte, and Gilles Venturini. Learning error-correcting graph matching with a multiclass neural network. *Pattern Recognition Letters*, 2018, in press. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2018.03.031. URL http://www.sciencedirect.com/science/article/pii/S0167865518301107

- Darwiche et al. [2018, in pressb] Mostafa Darwiche, Donatello Conte, Romain Raveaux, and Vincent T'Kindt. Graph edit distance: Accuracy of local branching from an application point of view. *Pattern Recognition Letters*, 2018, in pressb. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2018.03.033. URL http://www.sciencedirect.com/science/article/pii/S0167865518301119

- Darwiche et al. [2018, in pressa] Mostafa Darwiche, Donatello Conte, Romain Raveaux, and Vincent T'Kindt. A local branching heuristic for solving a graph edit distance problem. *Computers and Operations Research*, 2018, in pressa. ISSN 0305-0548. doi:https://doi.org/10.1016/j.cor.2018.02.002. URL http://www.sciencedirect.com/science/article/pii/S0305054818300339

- Abu-Aisheh et al. [2018] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. A parallel graph edit distance algorithm. *Expert Syst. Appl.*, 94:41–57, 2018. doi:10.1016/j.eswa.2017.10.043. URL https://doi.org/10.1016/j.eswa.2017.10.043

- Alaei et al. [2018] Alireza Alaei, Donatello Conte, Maxime Martineau, and Romain Raveaux. Blind document image quality prediction based on modification of quality aware clustering method integrating a patch selection strategy. *Expert Syst. Appl.*, 108:183–192, 2018. doi:10.1016/j.eswa.2018.05.007. URL https://doi.org/10.1016/j.eswa.2018.05.007

- Martineau et al. [2017] Maxime Martineau, Donatello Conte, Romain Raveaux, Ingrid Arnault, Damien Munier, and Gilles Venturini. A survey on image-based insect classification. *Pattern Recognition*, 65:273–284, 2017. doi:10.1016/j.patcog.2016.12.020. URL https://doi.org/10.1016/j.patcog.2016.12.020

- Lerouge et al. [2017] Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. New binary linear programming formulation to compute the graph edit distance. *Pattern Recognition*, 72:254–265, 2017. doi:10.1016/j.patcog.2017.07.029. URL https://doi.org/10.1016/j.patcog.2017.07.029

- Abu-Aisheh et al. [2017a] Zeina Abu-Aisheh, Benoit Gaüzère, Sébastien Bougleux, Jean-Yves Ramel, Luc Brun, Romain Raveaux, Pierre Héroux, and Sébastien Adam. Graph edit distance contest: Results and future challenges. *Pattern Recognition Letters*, 100:96–103, 2017a. doi:10.1016/j.patrec.2017.10.007. URL https://doi.org/10.1016/j.patrec.2017.10.007

- Alaei et al. [2017] Alireza Alaei, Romain Raveaux, and Donatello Conte. Image quality assessment based on regions of interest. *Signal, Image and Video Processing*, 11(4):673–680, 2017. doi:10.1007/s11760-016-1009-z. URL https://doi.org/10.1007/s11760-016-1009-z

- Abu-Aisheh et al. [2016] Zeina Abu-Aisheh, Romain Raveaux, and Jean-Yves Ramel. Anytime graph matching. *Pattern Recognition Letters*, 84:215–224, 2016. doi:10.1016/j.patrec.2016.10.004. URL https://doi.org/10.1016/j.patrec.2016.10.004

- Raveaux et al. [2013a] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. Structured representations in a content based image retrieval context. *J. Visual Communication and Image Representation*, 24(8):1252–1268, 2013a. doi:10.1016/j.jvcir.2013.08.010. URL https://doi.org/10.1016/j.jvcir.2013.08.010

- Raveaux et al. [2012] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A local evaluation of vectorized documents by means of polygon assignments and matching. *IJDAR*, 15(1):21–43, 2012. doi:10.1007/s10032-010-0143-3. URL https://doi.org/10.1007/s10032-010-0143-3

- Raveaux et al. [2011] Romain Raveaux, Sébastien Adam, Pierre Héroux, and Éric Trupin. Learning graph prototypes for shape recognition. *Computer Vision and Image Understanding*, 115(7):905–918, 2011. doi:10.1016/j.cviu.2010.12.015. URL https://doi.org/10.1016/j.cviu.2010.12.015

- Raveaux et al. [2010] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A graph matching method and a graph matching distance based on subgraph assignments. *Pattern Recognition Letters*, 31(5):394–406, 2010. doi:10.1016/j.patrec.2009.10.011. URL https://doi.org/10.1016/j.patrec.2009.10.011

E.1.4 Revues Nationales

- Leturcq and Raveaux [2016] Samuel Leturcq and Romain Raveaux. Les graphes pour étudier les dynamiques spatiales à partir des séries fiscales médiévales et modernes. Etat des lieux de l'expérience Modelespace. *Bulletin du Centre d'études médiévales d'Auxerre. Hors-série*, (9), April 2016. doi:10.4000/cem.13805. URL https://hal.archives-ouvertes.fr/hal-01311067

E.1.5 Conférences Internationales de rang A (CORE 2018)

- Lerouge et al. [2016] Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. Exact graph edit distance computation using a binary linear program. In Antonio Robles-Kelly, Marco Loog, Battista Biggio, Francisco Escolano, and Richard C. Wilson, editors, *Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop, S+SSPR 2016, Mérida, Mexico, November 29 - December 2, 2016, Proceedings*, volume 10029 of *Lecture Notes in Computer Science*, pages 485–495, 2016. ISBN 978-3-319-49054-0. doi:10.1007/978-3-319-49055-7_43. URL https://doi.org/10.1007/978-3-319-49055-7_43

- Alaei et al. [2015] Alireza Alaei, Donatello Conte, and Romain Raveaux. Document image quality assessment based on improved gradient magnitude similarity deviation. In *13th International Conference on Document Analysis and Recognition, ICDAR 2015, Nancy, France, August 23-26, 2015*, pages 176–180. IEEE Computer Society, 2015. ISBN 978-1-4799-1805-8. doi:10.1109/ICDAR.2015.7333747. URL https://doi.org/10.1109/ICDAR.2015.7333747

- Raveaux et al. [2007c] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A colour document interpretation: Application to ancient cadastral maps. In *9th International Conference on Document Analysis and Recognition (ICDAR 2007), 23-26 September, Curitiba, Paraná, Brazil*, pages 1128–1132. IEEE Computer Society, 2007c. ISBN 978-0-7695-2822-9. doi:10.1109/ICDAR.2007.5. URL http://doi.ieeecomputersociety.org/10.1109/ICDAR.2007.5

E.1.6 Conférences Internationales de rang B (CORE 2018)

- Alaei et al. [2016] Alireza Alaei, Donatello Conte, Michael Blumenstein, and Romain Raveaux. Document image quality assessment based on texture similarity index. In *12th IAPR Workshop on Document Analysis Systems, DAS 2016, Santorini, Greece, April 11-14, 2016*, pages 132–137. IEEE Computer Society, 2016. ISBN 978-1-5090-1792-8. doi:10.1109/DAS.2016.33. URL https://doi.org/10.1109/DAS.2016.33

- Coustaty et al. [2011] Mickaël Coustaty, Romain Raveaux, and Jean-Marc Ogier. Historical document analysis: A review of french projects and open issues. In *Proceedings of the 19th European Signal Processing Conference, EUSIPCO 2011, Barcelona, Spain, August 29 - Sept. 2, 2011*, pages 1445–1449. IEEE, 2011. URL http://ieeexplore.ieee.org/document/7074005/

- Raveaux et al. [2008a] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. Object extraction from colour cadastral maps. In Koichi Kise and Hiroshi Sako, editors, *The Eighth IAPR International Workshop on Document Analysis Systems, DAS 2008, September 16-19, 2008, Nara, Japan*, pages 506–514. IEEE Computer Society, 2008a. ISBN 978-0-7695-3337-7. doi:10.1109/DAS.2008.9. URL https://doi.org/10.1109/DAS.2008.9

- Raveaux et al. [2008b] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A colour text/graphics separation based on a graph representation. In *19th International Con-*

ference on Pattern Recognition (ICPR 2008), December 8-11, 2008, Tampa, Florida, USA, pages 1–4. IEEE Computer Society, 2008b. ISBN 978-1-4244-2175-6. doi:10.1109/ICPR.2008.4761725. URL https://doi.org/10.1109/ICPR.2008.4761725

- Barbu et al. [2006] Eugen Barbu, Romain Raveaux, Hervé Locteau, Sébastien Adam, Pierre Héroux, and Éric Trupin. Graph classification using genetic algorithm and graph probing application to symbol recognition. In *18th International Conference on Pattern Recognition (ICPR 2006), 20-24 August 2006, Hong Kong, China* DBL [2006], pages 296–299. ISBN 0-7695-2521-0. doi:10.1109/ICPR.2006.612. URL https://doi.org/10.1109/ICPR.2006.612

- Locteau et al. [2006] Hervé Locteau, Romain Raveaux, Sébastien Adam, Yves Lecourtier, Pierre Héroux, and Éric Trupin. Approximation of digital curves using a multi-objective genetic algorithm. In *18th International Conference on Pattern Recognition (ICPR 2006), 20-24 August 2006, Hong Kong, China* DBL [2006], pages 716–719. ISBN 0-7695-2521-0. doi:10.1109/ICPR.2006.276. URL https://doi.org/10.1109/ICPR.2006.276

E.1.7 Autres conférences Internationales

- Darwiche et al. [2017] Mostafa Darwiche, Romain Raveaux, Donatello Conte, and Vincent T'Kindt. A local branching heuristic for the graph edit distance problem. In Marcelo Mendoza and Sergio A. Velastin, editors, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications - 22nd Iberoamerican Congress, CIARP 2017, Valparaíso, Chile, November 7-10, 2017, Proceedings*, volume 10657 of *Lecture Notes in Computer Science*, pages 194–202. Springer, 2017. ISBN 978-3-319-75192-4. doi:10.1007/978-3-319-75193-1_24. URL https://doi.org/10.1007/978-3-319-75193-1_24

- Raveaux et al. [2017] Romain Raveaux, Maxime Martineau, Donatello Conte, and Gilles Venturini. Learning graph matching with a graph-based perceptron in a classification context. In Foggia et al. [2017], pages 49–58. ISBN 978-3-319-58960-2. doi:10.1007/978-3-319-58961-9_5. URL https://doi.org/10.1007/978-3-319-58961-9_5

- Abu-Aisheh et al. [2017b] Zeina Abu-Aisheh, Romain Raveaux, and Jean-Yves Ramel. Fast nearest neighbors search in graph space based on a branch-and-bound strategy. In Foggia et al. [2017], pages 197–207. ISBN 978-3-319-58960-2. doi:10.1007/978-3-319-58961-9_18. URL https://doi.org/10.1007/978-3-319-58961-9_18

- Abu-Aisheh et al. [2015a] Zeina Abu-Aisheh, Romain Raveaux, and Jean-Yves Ramel. A graph database repository and performance evaluation metrics for graph edit distance. In Cheng-Lin Liu, Bin Luo, Walter G. Kropatsch, and Jian Cheng, editors, *Graph-Based Representations in Pattern Recognition - 10th IAPR-TC-15 International Workshop, GbRPR 2015, Beijing, China, May 13-15, 2015. Proceedings*, volume 9069 of *Lecture Notes in Computer Science*, pages 138–147. Springer, 2015a. ISBN 978-3-319-18223-0. doi:10.1007/978-3-319-18224-7_14. URL https://doi.org/10.1007/978-3-319-18224-7_14

- Abu-Aisheh et al. [2015b] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In Maria De Marsico, Mário A. T. Figueiredo, and Ana L. N. Fred, editors, *ICPRAM 2015 - Proceedings of the International Conference on Pattern Recognition Applications and Methods, Volume 1, Lisbon, Portugal, 10-12 January, 2015.*, pages 271–278. SciTePress, 2015b. ISBN 978-989-758-076-5

- Valveny et al. [2011] Ernest Valveny, Mathieu Delalandre, Romain Raveaux, and Bart Lamiroy. Report on the symbol recognition and spotting contest. In Young-Bin Kwon and Jean-Marc Ogier, editors, *Graphics Recognition. New Trends and Challenges - 9th International*

Workshop, GREC 2011, Seoul, Korea, September 15-16, 2011, Revised Selected Papers, volume 7423 of *Lecture Notes in Computer Science*, pages 198–207. Springer, 2011. ISBN 978-3-642-36823-3. doi:10.1007/978-3-642-36824-0_19. URL https://doi.org/10.1007/978-3-642-36824-0_19

- Raveaux and Hillairet [2010] Romain Raveaux and Guillaume Hillairet. Model driven image segmentation using a genetic algorithm for structured data. In Manuel Graña Romay, Emilio Corchado, and M. Teresa García-Sebastían, editors, *Hybrid Artificial Intelligence Systems, 5th International Conference, HAIS 2010, San Sebastián, Spain, June 23-25, 2010. Proceedings, Part I*, volume 6076 of *Lecture Notes in Computer Science*, pages 311–318. Springer, 2010. ISBN 978-3-642-13768-6. doi:10.1007/978-3-642-13769-3_38. URL https://doi.org/10.1007/978-3-642-13769-3_38

- Raveaux et al. [2008c] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A colour space selection scheme dedicated to information retrieval tasks. In Alfons Juan-Císcar and Gemma Sánchez-Albaladejo, editors, *Pattern Recognition in Information Systems, Proceedings of the 8th International Workshop on Pattern Recognition in Information Systems, PRIS 2008, In conjunction with ICEIS 2008, Barcelona, Spain, June 2008*, pages 123–134. INSTICC PRESS, 2008c. ISBN 978-989-8111-42-5

- Raveaux et al. [2007a] Romain Raveaux, Eugen Barbu, Hervé Locteau, Sébastien Adam, Pierre Héroux, and Éric Trupin. A graph classification approach using a multi-objective genetic algorithm application to symbol recognition. In Francisco Escolano and Mario Vento, editors, *Graph-Based Representations in Pattern Recognition, 6th IAPR-TC-15 International Workshop, GbRPR 2007, Alicante, Spain, June 11-13, 2007, Proceedings*, volume 4538 of *Lecture Notes in Computer Science*, pages 361–370. Springer, 2007a. ISBN 978-3-540-72902-0. doi:10.1007/978-3-540-72903-7_33. URL https://doi.org/10.1007/978-3-540-72903-7_33

- Raveaux et al. [2007b] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A segmentation scheme based on a multi-graph representation: Application to colour cadastral maps. In Wenyin Liu, Josep Lladós, and Jean-Marc Ogier, editors, *Graphics Recognition. Recent Advances and New Opportunities, 7th International Workshop, GREC 2007, Curitiba, Brazil, September 20-21, 2007. Selected Papers*, volume 5046 of *Lecture Notes in Computer Science*, pages 202–212. Springer, 2007b. ISBN 978-3-540-88184-1. doi:10.1007/978-3-540-88188-9_20. URL https://doi.org/10.1007/978-3-540-88188-9_20

- Raveaux et al. [2007d] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A contribution to ancient cadastral maps interpretation through colour analysis. In Ana L. N. Fred and Anil K. Jain, editors, *Pattern Recognition in Information Systems, Proceedings of the 7th International Workshop on Pattern Recognition in Information Systems, PRIS 2007, In conjunction with ICEIS 2007, Funchal, Madeira, Portugal, June 2007*, pages 89–98. INSTICC PRESS, 2007d. ISBN 978-972-8865-93-1

- Locteau et al. [2005] Hervé Locteau, Romain Raveaux, Sébastien Adam, Yves Lecourtier, Pierre Héroux, and Éric Trupin. Polygonal approximation of digital curves using a multi-objective genetic algorithm. In Wenyin Liu and Josep Lladós, editors, *Graphics Recognition. Ten Years Review and Future Perspectives, 6th Internation Workshop, GREC 2005, Hong Kong, China, August 25-26, 2005, Revised Selected Papers*, volume 3926 of *Lecture Notes in Computer Science*, pages 300–311. Springer, 2005. ISBN 978-3-540-34711-8. doi:10.1007/11767978_27. URL https://doi.org/10.1007/11767978_27