

Contents

1	Graph Matching	2
1.1	Basic definitions and notation	2
1.2	Exact graph matching	4
1.3	Inexact graph matching	5
1.3.1	Graph Matching Objective Function	6
1.4	Approaches for solving graph matching problem	11
1.4.1	Discrete optimization	11
1.4.2	Continuous optimization	12
1.5	Graph matching algorithms studied in this thesis	16
1.6	Discussion	16
A	Quadratic Assignment Problem	17
B	Bibliography	20

Chapter 1

Graph Matching

In this chapter we review basic definitions and notations from the graph theory used in this thesis and introduce different forms and formulations of the graph matching problem together with some algorithms for solving them. The classification we use is based on [16], although we concentrated our attention mainly on one specific group of algorithms: those, which consider the graph matching problem as a quadratic optimization problem. Not all algorithms, we present thereby, were initially mentioned in [16], but we also do not cover all of the recent ones due to their quantity. We focus ourselves specially on those, that are important for further reading of the thesis.

1.1 Basic definitions and notation

A *undirected graph* $G = (V, E)$ is defined as a pair of disjoint sets V, E , where $E \subseteq \{\{u, v\} | u, v \in V\}$ [17]. The elements of the set V are called *vertices* or *nodes*¹ and the elements of E are called *edges*. Where it is necessary, we will write $V(G), E(G)$ to refer node and edge sets to the graph G .

The number of nodes in V defines the *size* of a graph G . Two nodes $v_i, v_{i'} \in V$ are called *adjacent*, if there is an edge $e = \{v_i, v_{i'}\} \in E$. Each graph can be represented by its *adjacency matrix* $A = (a_{ij})_{n \times n}$, where

$$a_{ij} = \begin{cases} 1, & \text{if } \{v_i, v_{i'}\} \in E, \\ 0, & \text{otherwise.} \end{cases}$$

and n is the number of nodes in the graph.

¹We use terms vertex and node further in the text as synonyms

A *path* in a graph $G = (V, E)$ is a sequence of nodes $\{v_0, v_1, \dots, v_k\}$ connected by the edges $\{v_0v_1, v_1v_2, \dots, v_{k-1}v_k\}$, where $v_i \in V$ and $v_{i-1}v_i \in E$ for all $i = 0, \dots, k$. A path with $v_0 = v_k$ is a *cycle*.

A graph $G' = (V', E')$ is called a *subgraph* of a graph $G = (V, E)$, if $V' \subseteq V$ and $E' \subseteq E$. We use the standard notation $G' \subseteq G$ for this. The subgraph G' of G is *induced by the set* $V' \subseteq V$, if $E' = \{(v_i, v_{i'}) | v_i, v_{i'} \in V'\}$. Analog, the set $E' \subseteq E$ induces the subgraph G' of G , if $V' = \{v \in V | v \in e \text{ and } e \in E'\}$. For the induced subgraph we use the notation $G' = G[V']$ and $G' = G[E']$ [17], depending if it is node- or vertex-induced. We also define graph cut $G \cap G' = (V \cap V', E \cap E')$ and union $G \cup G' = (V \cup V', E \cup E')$.

There are several special types of graphs. A graph, whose each pair of nodes is connected by an edge is called *complete*. In case, when each node $v_i \in V$ of a graph G has an associated attribute $d_i \in D$, one speaks about *attributed graph* $G = (V, E, D)$. If in contract to this each edge of a graph has an associated weight, the graph is called *weighted graph*. A connected, undirected graph without cycles is called a *tree*. A *hypergraph* is graph, whose edges connect several vertices at the same time (*hyperedges*).

Consider two undirected attributed graphs $G^I = (V^I, E^I, D^I)$ and $G^J = (V^J, E^J, D^J)$. We assume the situation, where $|V^I| = n_1$, $|V^J| = n_2$ and $n_1 \leq n_2$. A *matching function* between G^I and G^J is a mapping $m : V^I \rightarrow V^J$ between the sets of nodes of two graphs. It is clear, that defined in this way the mapping m is not unique. Assume, that we have some metric $S(G^I, G^J, m)$ to measure the quality of matching m . In this case, *Graph matching problem* between G^I and G^J can be defined as a problem of finding such a map $m : V^I \rightarrow V^J$, that maximizes the similarity score $S(G^I, G^J, m)$ between the graphs and fulfills some additional constraints:

$$m = \underset{\hat{m}}{\operatorname{argmax}} S(G^I, G^J, \hat{m}) \quad (1.1)$$

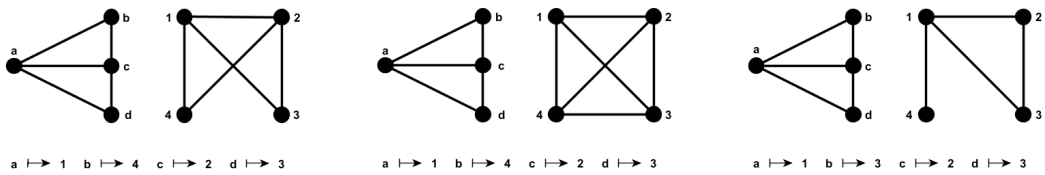
Based on the required properties of the mapping m the algorithms, that solve this problem, can be divided into two large groups [16]: *exact* and *inexact* graph matching methods. There are also variations inside of each group based on the definition of the similarity score between two graphs. In the following sections we will give an overview of the common exact and inexact graph matching algorithms.

1.2 Exact graph matching

The group of exact graph matching algorithms represents a class of more strict methods, that require the mapping between nodes of two graphs to be *edge preserving*. With other words: if $\{v_i, v_{i'}\} \in E^I$, then $\{m(v), m(v_{i'})\} \in E^J$ for all $v_i, v_{i'} \in V^I$. The graphs, considered in this case often do not have attributes.

There are several forms of the exact graph matching. The most known one is *graph isomorphism*: two graphs are called *isomorph* ($G^I \simeq G^J$), if the edge preserving mapping between their nodes is bijective. This implies, that two graphs have the equal number of nodes. If it is not the case, and the isomorphism holds between the one graph and a node-induced subgraph of the other graphs, the problem is called *subgraph isomorphism*. Its further extension is a isomorphism between subgraphs of the graphs. The last problem can obviously have several solutions, but one is normally interested in finding a common subgraph with maximum number of nodes or edges (*maximum common subgraphs*).

The further simplification of the graph isomorphism is to require an injective edge-preserving mapping, instead of bijective. This problem is called *graph monomorphism*. The correspondences here are still one-to-one, but the second graph may contain additional nodes and edges, comparing to the first graph. Even weaker form of graph matching problem is *graph homomorphism*. It allows many-to-one mapping between nodes of two graphs, which means that the only one restriction on the mapping m is to be total. On the Fig. 1.1 one can see examples of different exact graph matching problems.



(a) Graph isomorphism (b) Graph monomorphism (c) Graph homomorphism

Figure 1.1: Exact graph matching problems

All problems except graph isomorphism are proofed to be *NP*–complete [20]. This can be shown through the reduction of the matching problem to the clique problem. The graph isomorphism is currently shown to be *NP*–hard [20, 39]. For special

types of graphs there exist polynomial time algorithms (e.g. for trees [1, 20]).

The most widespread approach to solve exact graph matching problems is based on tree search with backtracking. The idea is to start with an empty solution and try in each step to expand it based on some rules until the solution is found. If it happens, that on some step a current solution cannot be expanded further due to problem's constraints, the current branch is cut and algorithm backtracks to the last feasible solution and tries to expand it in other way. Those algorithms are very slow, if they need to search through the whole tree, but can be speeded up by applying some heuristic to detect unpromising branches and exclude them from the search. The most known algorithm in this group, which uses depth-first-rule to traverse a tree, is the branch and bound [35] algorithm.

One of the first algorithms, that uses described techniques, is the one by Ullman [44]. Later it was extended and improved mostly by the suggestion of new pruning heuristics (see for example [25] for comparison of algorithms based on the same technique). We also want to mention here another well known algorithm, which uses the tree search method and is closely related to maximal common subgraph problem, - algorithm by Bron and Kerbosch [3] for finding cliques in an undirected graphs.

The other techniques, that were successfully applied for the (sub)graph isomorphism are decision trees [31, 42, 41] and group theory [30] for maximal subgraph matching.

1.3 Inexact graph matching

Sometimes it is difficult to apply the exact matching algorithms described in previous section to real world problems. There are two possible reasons for that. On the one hand, that are variations in the structure of graphs, that describe a same object. Those variations could be a consequence of natural deformations or noise influence, that can occur in some real world applications. On the other hand, solving the graph matching problem exactly can be time or memory consuming. As a consequence one can be interested in solving graph matching problem inexactly. In this case, no edge preserving mapping between the nodes of two graphs is required.

Let us recall the problem statement (1.1):

$$m = \operatorname{argmax}_{\hat{m}} S(G^I, G^J, \hat{m})$$

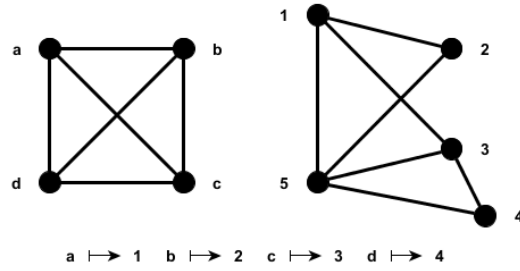


Figure 1.2: Inexact graph matching

where $S(G^I, G^J, \hat{m})$ defines the similarity measure between the attributed graphs $G^I = (V^I, E^I, D^I)$ and $G^J = (V^J, E^J, D^J)$. The mapping m is required to be total and sometimes also injective, to guarantee one-to-one matching.

Depending if an algorithm for solving (1.1) finds a global solution or not, it is called *optimal* or *suboptimal*. The choice, which algorithm to select depends on a specific problem. It should be noted, that an optimal inexact algorithms is not necessary faster than the exact one. On the other hand, suboptimal inexact algorithms often do not have any performance guarantee.

There are also different ways to define a similarity function $S(G^I, G^J, m)$, which leads to high number of different approaches inside the group of inexact graph matching algorithms. In the following section we summarized the most common form of objective function of the problem (1.1).

1.3.1 Graph Matching Objective Function

In some literature instead of defining the similarity function as in (1.1) one speaks about dissimilarity between two graphs [cite](#) or matching score [cite](#). The goal in this case is to minimize those values. Two versions of the problem formulation are however equivalent and one can easy transform one into other.

Quadratic Optimization Problem

The objective of graph matching is to find a mapping between two attributed graphs $G^I = (V^I, E^I, D^I)$ and $G^J = (V^J, E^J, D^J)$. Let the size of the graphs be n_1 and n_2 respectively. Generally, n_1 and n_2 can be different, but then we assume without losing generality $n_1 \leq n_2$. Here and further, we require the mapping m in (1.1) to

be total and injective, which guarantees, that each node of the first graph will be matched to exactly one node of the second graph (one-to-one matching).

We start with a even stricter assumption, that $n_1 = n_2 = n$ and the matching is bijective. In this case a mapping between nodes of the two graphs defines a permutation σ of the set $\{1, \dots, n\}$. Each permutation σ can be represented by the permutation matrix $P = \{P_{ij}\}$, where

$$P_{ij} = \begin{cases} 1, & \text{if } \sigma(i) = j, \\ 0, & \text{otherwise.} \end{cases}$$

We denote with Π_n the set of all feasible permutations:

$$\Pi_n = \{P \in \{0, 1\}^{n \times n} \mid \sum_{i=1, \dots, n} P_{ij} = \sum_{j=1, \dots, n} P_{ij} = 1 \quad \forall i, j = 1, \dots, n\}$$

Let matrices A^I and A^J be the adjacency matrices of the graphs G^I and G^J respectively. We assume, that in case of weighed graphs, the adjacency matrices contain edge weights, instead of binary values. We can now formulate the graph matching problem as (compare with [28, 37]cite):

$$P = \underset{\hat{P} \in \Pi_n}{\operatorname{argmin}} \|A^I - \hat{P}A^J\hat{P}^T\|^2 + \|D^I - \hat{P}D^J\|^2 \quad (1.2)$$

where $\|\cdot\|$ is the matrix Frobenius norm. Some authors [46] use slightly different, but equivalent reformulation of it, namely:

$$P = \underset{\hat{P} \in \Pi_n}{\operatorname{argmin}} \|A^I\hat{P} - \hat{P}A^J\|^2 + \|D^I - \hat{P}D^J\|^2, \quad (1.3)$$

After some transformations, the problem (1.2) can be reformulated as (see [8], Appendix A):

$$P = \underset{\hat{P} \in \Pi_n}{\operatorname{argmin}} \operatorname{vec}(\hat{P})^T (-(A^J)^T \otimes (A^I)^T) \operatorname{vec}(\hat{P}) + \operatorname{tr}(D\hat{P}^T) \quad (1.4)$$

where \otimes denotes the Kronecker product and $\operatorname{vec}(\hat{P})$ is a column-wise vectorization of $\hat{P} \in \Pi_n$.

The objective function of (1.4) is a negation of the objective of the *quadratic assignment problem*, which is know to be *NP*-hard [8, 38].

The both formulations have their advantages and disadvantages. The main benefit of (1.2) is the low space complexity $\mathcal{O}(n^2)$, where the space requirement of (1.4) estimates with $\mathcal{O}(n^4)$. This makes the second formulation tractable only for relative

small graphs. The drawback of both formulations is the strict penalization function of edge disagreements, namely the squared euclidean distance between matched edges. This follows straight forward from (1.2). In this sense the last formulation can be easily generalized in the way, we represent below.

We return back to the case where $n_1 \neq n_2$. Let denote with a binary vector $x \in \{0, 1\}^{n_1 n_2}$ the column-wise vectorization of the assignment matrix P , which is not necessary a permutation matrix anymore. It is obviously, that $x_{(j-1)n_1+i} = 1$, if a node $v_i \in V^I$ is matched to a node $u_j \in V^J$, and $x_{(j-1)n_1+i} = 0$ otherwise. For simplicity we will write further x_{ij} instead of complicated form $x_{(j-1)n_1+i}$.

To measure a similarity between graphs one consider two different kinds of similarities: second-order *edge similarity* and first-order *node similarity*. The first one is defined as a function of the edges $s_E : E^I \times E^J \rightarrow \mathbb{R}$ and should penalize disagreements in the structure of two graphs. The second one $s_V : V^I \times V^J \rightarrow \mathbb{R}$ represent additional constrains on the possible node correspondences.

Using the introduced notation and definitions of the node and edge similarity functions the function $S(G^I, G^J, m)$ in (1.1) can be now rewrite as follows:

$$S(G^I, G^J, m) = \sum_{\substack{x_{ij}=1 \\ x_{i'j'}=1}} s_E(e_{ij}, e_{i'j'}) + \sum_{x_{ij}=1} s_V(v_i, v_j) \quad (1.5)$$

This formula can be also expressed in matrix form. We define an *affinity* or *similarity matrix* $A \in \mathbb{R}^{n_1 n_2 \times n_1 n_2}$, whose diagonal elements are $s_V(v_i, u_j)$ and non-diagonal elements are $s_E(e_{ii'}, e_{jj'})$. Using this matrix we become the following formulation of the graph matching problem as an quadratic optimization problem [10, 12, 13, 16, 27]:

$$\operatorname{argmax}_x x^T S x \quad (1.6)$$

$$\text{s.t. } x \in \{0, 1\}^{n_1 n_2} \quad (1.7)$$

$$\sum_{i=1 \dots n_1} x_{ij} \leq 1 \quad (1.8)$$

$$\sum_{j=1 \dots n_2} x_{ij} \leq 1 \quad (1.9)$$

We notice, that in the case, where $n_1 = n_2$, both conditions (1.8) and (1.9) will be fulfilled with equality.

One can easily see, that the formulation (1.4) can be obtained from (1.2)-(1.9) by

setting $S = -(A^J)^T \otimes (A^I)^T$.

Below we list a different ways to define a node and edge similarities between two attributed graphs we found in the literature.

- Node similarity

The most frequently used approach to define a node similarity is to compare attributes of the corresponding nodes. In most of seen literature, the node attributes are represented by d -dimensional real vectors : $D^i, D^J \subset \mathbb{R}^d$. One of the simplest ways to define a node similarity in this case is to use a distance metric:

$$s_V(v_i, v_j) = \text{dist}(d_i, d_j), \text{ for } \forall v_i \in V^I, \forall v_j \in V^J \quad (1.10)$$

It can be, for example, an euclidean distance ([12]cite). The other widely used alternative to define node similarity, is to use cosine similarity² of node attributes [33]cite: $\text{dist}_c(d_i, d_j) = \frac{d_i \cdot d_j}{\|d_i\|_2 \|d_j\|_2}$, where dot denotes scalar product of two vectors.

- Edge similarity

One possible approach to calculate edge similarity is to calculate *edge dissimilarity* $d_E : E^I \times E^J \rightarrow \mathbb{R}$ first and then transform it into similarity by using, for example, one of the following functions [10, 11, 12, 13]:

$$\begin{aligned} - s_E(e_{ii'}, e_{jj'}) &= \exp\left(-\frac{d(e_{ii'}, e_{jj'})^2}{\sigma_s^2}\right) \\ - s_E(e_{ii'}, e_{jj'}) &= \max(\beta - d(e_{ii'}, e_{jj'})/\sigma_s^2, 0) \end{aligned}$$

where $e_{ii'} \in E^I$ and $e_{jj'} \in E^J$. The parameters σ_s and β define a sensibility of a graph matching algorithm to the dissimilarities between the graphs.

This leads us to the question how to calculate edge dissimilarity. The most obvious way is to compare a weights of the edges, if such are provided. An other alternative could be to use length of the edges [10, 11, 12, 13], if coordinates of the nodes in some system (usually, Cartesian coordinates) are known. It a significant assumptions, which holds however for almost all graphs arose in practical application.

If the nodes of the graphs is described not only by their location, but also by some affine region around the nodes, one can use so-called geometric dissimi-

²Cosine similarity is not distance metric, because it does not fulfill the triangular inequality.

larity [11, 13]:

$$d(e_{ij}, e_{i'j'}) = \frac{1}{2}(d_{geo}(m_j|m_i) + d_{geo}(m_i|m_j)) \quad (1.11)$$

$$d_{geo}(m_j|m_i) = \frac{1}{2}(\|x_{j'} - H_i x_j\| + \|x_j - H_i^{-1} x_{j'}\|) \quad (1.12)$$

$$d_{geo}(m_i|m_j) = \frac{1}{2}(\|x_{i'} - H_j x_i\| + \|x_i - H_j^{-1} x_{i'}\|) \quad (1.13)$$

where m_i is a correspondence between nodes v_i and $v_{i'}$ and H_i is a homography from v_i to $v_{i'}$ defined by the provided affine regions around each node.

Error correcting graph matching

Another way to measure the similarity between two graphs is based on *graph edit distance* [6]. The graph edit distance is defined through costs of *graph edit operations*, that transform on graph into another. Those operations are insertion, deletion and substitution of the both nodes and edges. The algorithms, that uses graph edit distance for graph matching, are often called *error correcting* [16].

Consider two attributed graphs $G^I = (V^I, E^I, D^I)$ and $G^J = (V^J, E^J, D^J)$ and matching m between subsets \bar{V}^I, \bar{V}^J of V^I and V^J respectively. The edit operations on nodes can be directly defined by the mapping m [4]:

- if $m(v_i) = v_j$ for $v_i \in \bar{V}^I, v_j \in \bar{V}^J$, then a node v_i is *substituted* by a node v_j
- nodes in $V^I \setminus \bar{V}^I$ are *deleted*
- nodes in $V^J \setminus \bar{V}^J$ are *inserted*.

An edit operation of an edge is defined based on the transformations applied to it's end nodes. For example, if a node $v_j \in V^J$ was inserted, then all edges incident to this node are also inserted. Alternatively, if a node $v_i \in V^I$ was deleted, then all edges incident to this node are also deleted. We say, that an edge $\{v_i, v_{i'}\} \in E^I$ was substituted by an edge $\{v_j, v_{j'}\} \in E^J$, if the nodes $v_i, v_{i'}$ were mapped in $v_j, v_{j'}$.

We assume, that all operations can be performed simultaneously. Let $S = \{s_1, s_2, \dots, s_k\}$ denote a set of operations needed to transform the graph G^I into the graph G^J . Each edit operation $s_i, 1 \leq i \leq k$, has an assigned nonnegative cost $c(s_i)$. The cost of the whole sequence S is defined then as $\sum_{i=1}^k c(s_i)$. Using this one define a *edit distance* between the graphs G^I and G^J as follows [6, 47]:

$$dist(G^I, G^J, m) = \min_{S=\{s_1, \dots, s_k\}} c(S) \quad (1.14)$$

The cost functions are often defined as a functions of edge weights and node attributes. Their exact definitions depend however on an application. Sometimes, it can helpful, when the distance measure (1.14) defines a metric, which is not automatically the case. A list of restrictions on the cost functions, that ensure the distance (1.14) to be a metric, is provide i.e. in [6]. The same author in [7] provides a direct formulation of an edit distance measure, with metric properties without direct specification of cost functions.

An interesting question is, how does the error corrected graph matching problem correspond to the other matching problems. Some authors have shown, that some particular constrains on the graph edit costs allow to transform the first problem to the other or the other way round. In [5] it was shown, that graph isomorphism, subgraph isomorphism and maximum common subgraph problems can be considered as a special cases of the error correcting graph matching.

It is also easy to show, how to reformulate (1.14) into (1.2). The objective function of the last formulation can be rewritten as $\sum_{i=1}^n \sum_{j=1}^n (A_{ii'}^I - A_{\sigma(i)\sigma(i')}^J)^2$. Comparison of the last expression with the definition (1.14) leads us direct to the idea, how to select the edge insertion/deletion/substitution cost functions, to make the problem formulations equivalent:

$$c(e_{\text{subst}}) = (A_{ii'}^I - A_{\sigma(i)\sigma(i')}^J)^2 \quad c(e_{\text{insert}}) = (A_{\sigma(i)\sigma(i')}^J)^2 \quad c(e_{\text{del}}) = (A_{ii'}^I)^2$$

1.4 Approaches for solving graph matching problem

In following we briefly describe a common approaches for solving inexact graph matching problem, used in field of Computer Vision and Pattern Recognition. We subdivided them into groups based on their main idea.

1.4.1 Discrete optimization

Tree search methods

Similar to the exact graph matching problems the tree based methods with backtracking were successfully applied also to inexact matching problems. They were especially wild used for the error-correcting graph matching. The searching procedure often can be guided, so that the required time is shorter, than exponential

time, which is needed by a blind searching. To traverse a tree effectively one uses score of the current achieved partial matching together with a heuristic estimate for the score of the remaining nodes. The most algorithms in this groups differ in a suggested heuristics to estimate future costs and in a rules, for tree traversing. The most used methods for the last are depth-first-search and A^* -Algorithm [21]. The first algorithm for inexact graph matching based on the three search was presented in [43]. It is an optimal inexact graph matching algorithm. For further examples of the three search methods for inexact graph matching we refer to [6, 40, 47].

Simulated Annealing

Simulated annealing is a heuristic for searching the global optimum of a given energy function [8]. It is an extension of the Metropolis Algorithm [32], that was suggesting for finding an equilibrium of a physic system with many particles. Speaking in terms of a combinatorial optimization problem, feasible solutions of a problem define states of a system and corresponding objective scores it's energy. The idea is find the state with the lowest energy by performing some random changes in a system. A change, that pushes system into a state with lower energy, is always accepted. On the other side, a change, that increases energy function, is accepted with some probability. This probability and number of changes per iteration is controlled by the temperature parameter T . The bigger T the viewer changes are allowed and the smaller the accepted probability is.

The application of the simulated annealing to the graph matching problems can be found in [22]. It is also a common used heuristic used for quadratic assignment problem [8].

1.4.2 Continuous optimization

This group is one of the biggest and deep investigated. The main idea is to relax the integer constraints in the discrete optimization problems (1.2), (1.6) and apply continuous optimization algorithms to solve them. The found solution must be converted afterwards back in the discrete domain. We list further some of the techniques applied for solving relaxed problems.

The first subgroup of the algorithms, that use techniques of the continuous optimization, are those, who work directly with the objective function of (1.2) or (1.6). In 1996 Gold and Rangarajan presented a Graduated assignment algorithm for graph

matching (GAGM) [14]. It is an iterative algorithm, which uses *deterministic annealing*³ to solve the graph matching problem formulated as (1.6). They use Taylor series approximation to relax the initial quadratic assignment problem to linear assignment problem, which is then solved by using *soft-assign*. The basic idea of the soft-assigned is to transform a binary correspondence matrix to a doubly stochastic one. The deterministic annealing was also used in [34] to solve Lagrangian Relaxation of the problem (1.2) and in the robust point matching algorithm with thin-plane spline (RPM-TPS) [15]. The solution, obtained by this method is however not necessary optimal and the behavior of the algorithms depends highly on the selected parameters, especially those, which control the annealing schema.

Another iterative algorithm was proposed in [27] and is called an integer projected fixed point method (IPFP). It consists of two steps, which we shortly describe below. In the first step, an initial solution has to be chosen. It can be, for example, a continuous or discrete solution obtained by some other algorithm. The second step iteratively improves this solution by applying *Frank-Wolfe algorithm* [19] adapted to the graph matching problem. This algorithm finds first the best search direction by solving an linear assignment problem, which arises by the minimizing the first-order Taylor series approximation to the objective function. The minimization is performed in a discrete using the Hungarian algorithm [24]. Then the initial objective function is further maximized in a continuous domain along found direction. The authors show, that in the praxis IPFP tends to converge towards discrete solutions, that are close to the optimum.

The Fast Approximative Quadratic Programming Algorithm (FAQ), presented in [46], also uses Frank-Wolfe algorithm to solve a continuous relaxation of the problem (1.3). However FAQ perform completely in a continuous domain and has the third step, which maps a found continuous solution back onto discrete domain. Avoiding the usage of the similarity matrix S gives the FAQ a big advantage of the less space requirement, which make it possibly to apply the algorithm to graphs of a big size.

A similar approach is also used in the PATH-Algorithm [48]. The algorithm finds first the global optimum of the Lagrangian relaxation of the problem (1.3). For this the Newton method [2] is used. Afterwards the found solution is projected back onto discrete domain by solving a sequence of convex and conclave problems. Those

³Deterministic annealing is similar to the simulated annealing, but uses deterministic techniques to find the closest local minimum by value of the temperature parameter [36].

problems are solved again with Frank-Wolfe algorithm.

Very close to [27] is also the fast projected fixed-point algorithm (FastPFP) proposed in [28], that uses novel *projected fix point method* to find a solution of a continuous relaxation of the problem (1.2). Compared to IPFP the new algorithm uses projections onto a continuous domain, instead of discrete. Furthermore, the authors proofed the linear convergence rate of their algorithm, whereby the convergence rate of IPFP is unknown. The FastPFP, similar to FAQ, do not have a memory issues because of the other problem formulation and is generally faster than tow last mentioned methods, because it does not need to solve linear assignment problem on each iteration.

Spectral methods

A big group of algorithm for solving graph matching problem is represented by those, which use a eigenvalue decomposition [2] of the adjacency matrices of the graph, that should be matched. The idea behind this is, that the eigenvalues and eigenvector of the adjacency matrices of two isomorphic graphs are equal.

One of the first algorithms, which uses spectral techniques, is the one by Umeyama [45]. He considers the case, when a two graphs have the same number of nodes and matching between node of those graphs is bijective. This means, that a correspondence matrix must be a permutation matrix. He also assumes, that two graph are isomorphic or nearly isomorphic. The graph matching problem is then solved by defining an orthogonal matrix, which minimizes the objective score, based on the eigenvalue decomposition of the adjacency matrices of a given graphs. This algorithm is highly limited in it's application due to its requirements. We also want to mention, that it works only with graph geometry and does not consider attributes of nodes and edges.

The more rescent algorithm is this group of methods is called spectral matching (SM) [26]. The algorithm works with the general formulation (1.6) and considers the matrix S as an adjacency matrix of a weighted graph, whose nodes are define by all possible correspondences between the nodes of the original graphs. The matching problem reduces to the problem of finding a subgraph of this graphs with the maximum weight⁴. This problem can be formulated as an integer optimization problem. However the authors relax both mapping and integer constrains and obtain the op-

⁴A weight of a graph is defined here as a sum of weights of it edges.

timal solution of this relaxation by taken the principal eigenvector of the matrix S . An a discrete solution is obtained from the continuous by simple greedy heuristic.

A generalization of the SM algorithm is proposed in [18]. The authors suggest to consider similarities between tuples of nodes and not only pairs to improve graph matching quality. They formulate new problem using hyper-graphs and tensor notation. The solution strategy is afterwards the same, as in [26]: solution is approximated with the principal eigenvector of the matrix S . However, due to the increased complexity of the problem, the *the power iteration method* used in SM cannot be applied straight forward. So, the authors describe an effective way, how to apply existing method to the new problem.

Probabilistic approaches

Methods using clustering techniques

We want to emphasize independently the methods, that use clustering techniques to improve the graph matching score or performance.

Minsu Cho in [11] proposes to use *agglomerative clustering* on a set of candidate matches to effectively cop with outliers. The problem is formulates as in (1.6). The considered graphs are attributed, so that candidate matches are selected based on the distance between node attributes. The algorithm is controlled by the dissimilarity measure between two clusters, which is defined as the average of k smallest pairwise dissimilarities between points in two clusters. This linkage model is called by the authors *adaptive partial linkage model*. Notice, that the points in a clusters are pairwise correspondences between graph nodes. The dissimilarity between two nodes is defined as a linear combination of the node attribute dissimilarities and geometric dissimilarities defined as (1.11). At the end, the close correspondences build a bigger cluster and likely represent inliers. In contrast to that, the outliers are likely to stay in a small groups, so we can threshold them base on the cluster size.

Another algorithm based on the hierarchical clustering is described in [9]. Its propose is to provide additional constraints, that can be used to improve the robustness of a graph matching algorithm against graph deformations. This was successfully achieved by using spectral methods to build the clusters. The S eigenvectors associated with the first S largest eigenvalues are selected as a cluster centers. After performing the clustering, spectral matching algorithm is applied to match first the clusters and then the nodes inside the clusters.

Another benefit of using clustering method in graph matching is shown in [29]. The aim of the authors is to parallelize a graph matching algorithm for matching very big graphs. However the considered problem is semi supervised, meaning that correct correspondences between some nodes are known. It is obvious, that in the perfect clustering, node that should be matched, must lie in the corresponding subgraphs. To ensure, that the clustering is sufficiently good, the authors spectrally embed and then jointly cluster nodes of two graphs. The algorithm was successfully applied to the graphs with 20000 – 30000 nodes.

Recent algorithms for big graphs

1.5 Graph matching algorithms studied in this thesis

1.6 Discussion

Appendix A

Quadratic Assignment Problem

Consider a problem of assignment of n facilities to n locations given the transportation costs between the locations depending on the flow between them and opening costs of facilities in certain locations. The aim is to minimize the summary cost of the assignment. Let $D = (d_{kl}), F = (f_{kl}), B = (b_{ik}) \in \mathbb{R}^{n \times n}$ be a real matrices that define a distances and flow between the locations, as well as opening costs. The problem defined above can then be formulated as an integer quadratic program [8, 23]:

$$P = \operatorname{argmin}_{\hat{\sigma} \in \Sigma_n} \sum_{i=1}^n \sum_{j=1}^n f_{ij} d_{\hat{\sigma}(i)\hat{\sigma}(j)} + \sum_{i=1}^n b_{i\hat{\sigma}(i)}, \quad (\text{A.1})$$

where Σ_n is a set of all possible permutations of the set $\{1, \dots, n\}$, and is called *Koopmans-Beckmann version of the quadratic assignment problem* (further *QAP*).

We can assign a permutation matrix $P = (P_{ij}) \in \{0, 1\}^{n \times n}$ to each permutation σ , where $P_{i\sigma(i)} = 1$ and 0 elsewhere. The set of all feasible permutation matrices is defined as

$$\Pi_n = \{P \in \{0, 1\}^{n \times n} \mid \sum_{i=1}^n P_{ij} = \sum_{j=1}^n P_{ij} = 1 \quad \forall i, j = 1, \dots, n\}.$$

It is easy to see that, the formulation (A.1) is equivalent to

$$P = \operatorname{argmin}_{\hat{P} \in \Pi_n} (F \cdot \hat{P} D \hat{P}^T) + B \cdot \hat{P}, \quad (\text{A.2})$$

where \cdot denotes the Frobenius inner product of two square matrices defined as $A \cdot B = \sum_{i=1}^n \sum_{j=1}^n A_{ij} B_{ij}$.

We recall shortly the definition of the Kronecker product of two matrices and it's connection with the Frobenius inner product of two matrices and matrix trace.

Appendix A Quadratic Assignment Problem

The Kronecker product of two matrices $A = \{A_{ij}\}, B = \{B_{ij}\} \in \mathbb{R}^{n,n}$ is a new $n^2 \times n^2$ matrix C

$$C = A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{n1}B \\ \vdots & \ddots & \vdots \\ a_{1n}B & \dots & a_{nn}B \end{pmatrix}. \quad (\text{A.3})$$

From the definition of the matrix trace follows [cite](#):

$$A \cdot B = \sum_{i=1}^n \sum_{j=1}^n A_{ij} B_{ij} = \sum_{i=1}^n (AB^T)_{ii} = \text{tr}(AB^T) = \text{vec}(A)^T \text{vec}(B), \quad (\text{A.4})$$

where $\text{vec}(A)$ denotes the column-wise vectorization of a matrix A . Additionally, it holds [cite](#):

$$\text{vec}(APB) = (B^T \otimes A) \text{vec}(P). \quad (\text{A.5})$$

We can now apply the equality 2 and 3 in (A.4) to the formulation (A.2) and get the *trace formulation of QAP* [\[8\]](#):

$$P = \underset{\hat{P} \in \Pi_n}{\text{argmin}} \text{tr}(F\hat{P}D^T\hat{P}^T + B\hat{P}^T). \quad (\text{A.6})$$

The objective function of (A.6) can be further rewritten based on the last equality in (A.4) and property (A.5) as follows:

$$\begin{aligned} \text{tr}(F\hat{P}D^T\hat{P}^T + B\hat{P}^T) &= \text{tr}(\hat{P}(F\hat{P}D^T)^T) + \text{vec}(B)^T \text{vec}(\hat{P}) \\ &= \text{vec}(\hat{P})^T \text{vec}(F\hat{P}D^T) + \text{vec}(B)^T \text{vec}(\hat{P}) \\ &= \text{vec}(\hat{P})^T (D \otimes F) \text{vec} \hat{P} + \text{vec}(B)^T \text{vec}(\hat{P}). \end{aligned} \quad (\text{A.7})$$

Based on this formulation the matrix B is called sometimes *linear cost matrix* and the matrix $D \otimes F$ the *quadratic costs matrix*. If we assume the matrix B be a constant matrix with the same values, that it does not have any influence on the optimization process. In this case setting matrix F to $-(A^I)^T$ and matrix D to $(A^J)^T$ lead us to the formulation (1.3). Similarly, denoting the Kronecker product $(-D \otimes F)$ as S results in the formulation (1.5).

It remains to show, that (1.2) is equivalent to (A.6). Indeed, consider the objective function in (1.2):

$$\begin{aligned} \|A^I - \hat{P}A^J\hat{P}^T\|^2 &= \text{tr}((A^I - \hat{P}A^J\hat{P}^T)^T(A^I - \hat{P}A^J\hat{P}^T)) \\ &= \text{tr}(((A^I)^T - \hat{P}(A^J)^T\hat{P}^T)(A^I - \hat{P}A^J\hat{P}^T)) \\ &= \text{tr}((A^I)^T A^I - (A^I)^T \hat{P}A^J\hat{P}^T - \hat{P}(A^J)^T\hat{P}^T A^I + \hat{P}(A^J)^T\hat{P}^T \hat{P}A^J\hat{P}^T) \end{aligned}$$

$$= \text{tr}((A^I)^T A^I - 2(A^I)^T \hat{P} A^J \hat{P}^T + A^J (A^J)^T).$$

The first and the last term are obviously constant, and thus can be ignored in optimization. Finally, substitution of $F = -(A^I)^T$, $D = (A^J)^T$ leads us to the formulation (A.6). That is exactly, what we wanted to show.

Appendix B

Bibliography

- [1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.
- [2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.
- [3] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.
- [4] H. Bunke. Error-Tolerant Graph Matching: A Formal Framework and Algorithms. In A. Amin, D. Dori, P. Pudil, and H. Freeman, editors, *Advances in Pattern Recognition*, pages 1–14. Springer Berlin Heidelberg, 1998.
- [5] H. Bunke. Error correcting graph matching: on the influence of the underlying cost function. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):917–922, 1999.
- [6] H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, 1983.
- [7] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259, 1998.
- [8] R. E. Burkard, E. Çela, P. M. Pardalos, and L. Pitsoulis. The quadratic assignment problem. In P. M. Pardalos and D.-Z. Du, editors, *Handbook of Combinatorial Optimization*, pages 241–338. Kluwer Academic Publisher, 1998.
- [9] M. Carcassoni and E. R. Hancock. Correspondence matching with modal clusters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(12):1609–1615, 2003.
- [10] M. Cho and O. Duchenne. Finding Matches in a Haystack : A Max-Pooling

- Strategy for Graph Matching in the Presence of Outliers. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [11] M. Cho, J. Lee, and K. M. Lee. Feature Correspondence and Deformable Object Matching via Agglomerative Correspondence Clustering. In *The IEEE International Conference on Computer Vision (ICCV)*, 2009.
 - [12] M. Cho, J. Lee, and K. M. Lee. Reweighted Random Walks for Graph Matching. *ECCV*, 2010.
 - [13] M. Cho and K. M. Lee. Progressive graph matching: Making a move of graphs via probabilistic voting. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12. IEEE Computer Society, 2012.
 - [14] H. Chui and A. Rangarajan. A Graduated assignment algorithm for graph matching. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 18, pages 377–388, 1996.
 - [15] H. Chui and A. Rangarajan. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, 89:114–141, 2003.
 - [16] 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.
 - [17] R. Diestel. *Graph Theory, Electronic Edition 2005*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag New York, 2005.
 - [18] O. Duchenne, F. Bach, I.-s. Kweon, and J. Ponce. A Tensor-Based Algorithm for High-Order Graph Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(12):2383–2395, 2011.
 - [19] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.
 - [20] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.
 - [21] P. E. Hart, N. J. Nilsson, and B. Raphael. Formal Basis for the Heuristic Determination of Minimum Cost Path. *IEEE Transactions of Systems Science and Cybernetics*, 4(2):100–107, 1968.
 - [22] L. H'erault, R. Horaud, F. Veillon, and J. Niez. Symbolic image matching by simulated annealing. In *Proceedings of the British Machine Vision Conference*,

- pages 319–324, 1990.
- [23] T. Koopmans and M. J. Beckmann. Assignment problems and the location of economic activities. Cowles Foundation Discussion Papers 4, Cowles Foundation for Research in Economics, Yale University, 1955.
- [24] H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
- [25] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB’13, pages 133–144. VLDB Endowment, 2013.
- [26] M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. In *ICCV*, 2005.
- [27] M. Leordeanu, M. Hebert, R. Sukthankar, and M. Herbert. An Integer Projected Fixed Point Method for Graph Matching and MAP Inference. In *NIPS*, 2009.
- [28] Y. Lu, K. Huang, and C.-L. Liu. A fast projected fixed-point algorithm for large graph matching, 2012.
- [29] V. Lyzinski, D. L. Sussman, D. E. Fishkind, H. Pao, L. Chen, J. T. Vogelstein, Y. Park, and C. E. Priebe. Spectral Clustering for Divide-and-Conquer Graph Matching. 2011/14.
- [30] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *CoRR*, 2013.
- [31] B. Messmer and H. Bunke. A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recognition*, 32(12):1979–1998, 1999.
- [32] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.
- [33] W.-Z. Nie, A.-A. Liu, Z. Gao, and Y.-T. Su. Clique-graph Matching by Preserving Global & Local Structure. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [34] A. Rangarajan and E. Mjolsness. A lagrangian relaxation network for graph matching. In *IEEE Trans. Neural Networks*, pages 4629–4634. IEEE Press, 1996.

- [35] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice Hall College Div, 1977.
- [36] K. Rose. Deterministic annealing, clustering and optimization, 1991.
- [37] S. Roth. Analysis of a Deterministic Annealing Method for Graph Matching and Quadratic Assignment Problems in Computer Vision. Master’s thesis, University of Mannheim, 2001.
- [38] S. Sahni. Computationally Related Problems. *SIAM J. Comput*, 3(4):262–279, 1974.
- [39] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.
- [40] L. G. Shapiro and R. M. Haralick. Structural descriptions and inexact matching. *IEEE transactions on pattern analysis and machine intelligence*, 3(5):504–519, 1981.
- [41] K. Shearer, H. Bunke, and S. Venkatesh. Video sequence matching via decision tree path following. *Pattern recognition letters*, pages 479–492, 2001.
- [42] K. Shearer, H. Bunke, S. Venkatesh, and D. Kieronska. Efficient graph matching for video indexing. In *Graph based representations in pattern recognition*, Computing supplementum, pages 53–62. Springer-Verlag, 1998.
- [43] W.-H. Tsai and K.-S. Fu. Error-Correcting Isomorphisms of Attributed Relational Graphs for Pattern Analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(12):757–768, 1979.
- [44] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.
- [45] S. Umeyam. An Eigendecomposition Approach to Weighted Graph Matching Problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10, 1988.
- [46] J. T. Vogelstein, J. M. Conroy, L. J. Podrazik, S. G. Kratzer, E. T. Harley, D. E. Fishkind, R. J. Vogelstein, and C. E. Priebe. Large (Brain) Graph Matching via Fast Approximate Quadratic Programming, 2011/14.
- [47] J. T. Wang, K. Zhang, and G.-W. Chirn. Algorithms for approximate graph matching. *Information Sciences*, 82(1-2):45–74, 1995.
- [48] M. Zaslavskiy. Graph matching and its application in computer vision and

bioinformatics, 2010.

Declaration of Authorship

I confirm that this Master's thesis is my own work and I have documented all sources and material used. This thesis was not previously presented to another examination board and has not been published.

Heidelberg, den (Datum)