Contents

Chapter 1

Graph Matching

In this chapter we introduce different forms and formulations of the graph matching problem together with some algorithms for solving them. The classification we use is based on the one introduced by Conte et al. [18]. Not all algorithms, we will present, were initially mentioned in [18], but we also do not cover all of the resent ones due to their quantity. Our focus lies specially on those, that are important for further reading of the thesis.

To begin with we refresh basic definitions and notations from the graph theory used in this thesis.

1.1 Basic definitions and notations

An *undirected graph* $G = (V, E)$ is defined as a pair of disjoint sets V, E, where $E \subseteq \{\{u, v\} | u, v \in V\}$ [21]. 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 particular 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.

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

A *path* in a graph $G = (V, E)$ is a sequence of nodes $\{v_0, v_1, \ldots, v_k\}$ connected by the edges $\{v_0v_1, v_1v_2, \ldots, v_{k-1}v_k\}$, where $v_i \in V$ and $v_{i-1}v_i \in E$ for all $i = 1, \ldots, k$. 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. A subgraph G' of G is *induced by a node subsset* $V' \subseteq V$, if $E' = \{(v_i, v_{i'})|v_i, v_{i'} \in V'\}$. Analog, a subset $E' \subseteq E$ induces a subgraph G' of G, if $V' = \{v \in V|v \in e \text{ and } e \in E'\}$. For an induced subgraph we use the notation $G' = G[V']$ and $G' = G[E']$ [21], if it is node- or vertex-induced respectively. We also introduce 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 $m : V^I \to V^J$ 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 function $S(G^I, G^J, m)$ 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 $m : V^I \to V^J$, that maximizes the similarity score $S(G^I, G^J, m)$ between the graphs and fulfills some additional constraints:

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

Based on the required properties of the mapping m the algorithms, that solve this problem, can be divided into two large groups [18]: *exact* and *inexact* graph matching methods. There are also variations inside of each group based on the definition of the similarity function between two graphs. In the following sections we will give an overview of a common exact and inexact graph matching problems together with algorithms for solving them.

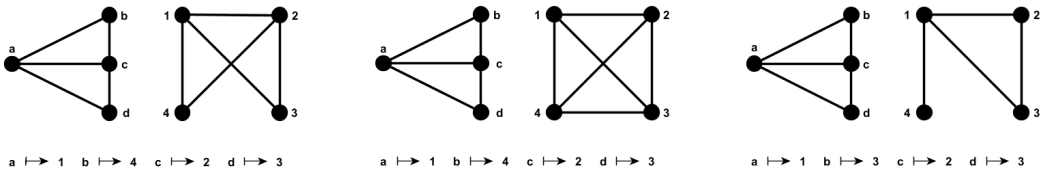
1.2 Exact graph matching

The group of exact graph matching algorithms represents a class of more strict methods, that require a mapping m 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 a edge preserving mapping between their nodes is bijective. This implies automatically, that two graphs should 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 graph, the problem is called *subgraph isomorphism*. Its further extension is the isomorphism between subgraphs of a 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*. A correspondences between nodes of a graphs are still one-to-one, but the second graph may contain additional nodes and edges, comparing to the first graph. Note, that each subgraph isomorphism defines a monomorphism on the whole graphs, however the opposite statement is not correct.

Even weaker form of a graph matching problem is *graph homomorphism*. It allows many-to-one mapping between nodes of two graphs, meaning that one node can correspond to several nodes of the other graph. The only one restriction on a mapping m in this case 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 [26]. This can be shown through a reduction of the respective matching problem to the clique problem. The graph isomorphism is currently shown to be $NP-$hard [26, 53]. For some special types of graphs there exist however polynomial time algorithms (e.g. for trees [1, 26]).

1.3 Methods for solving exact graph matching problems

The most widespread approach to solve an exact graph matching problem is based on a tree search with backtracking. This methods starts with an empty solution and tries to expend it in each step according to provided rules until the solution is found. Each solution represent a node of a tree. If it happens, that in some step a current solution cannot be expended further due to problem's constraints, the current branch in the tree is cut. The 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 [47] algorithm.

The one of the first algorithms, that uses described technique, is the one by Ullman [59]. Later it was extended and improved mostly by the suggestion of a new pruning heuristics (see for example [34] 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 [4] for finding cliques in an undirected graphs.

The other techniques, that were successfully applied for the (sub)graph isomorphism are decision trees [41, 56, 55] and group theory[40] for maximal subgraph matching.

1.4 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 word 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.

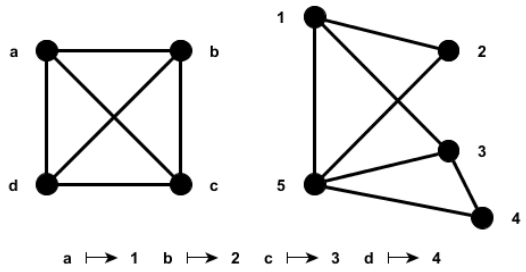


Figure 1.2: Inexact graph matching

Let us recall the problem statement (1.1):

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

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).

1.4.1 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 $\{1, \ldots, n\}$. 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:

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

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 [37, 50]cite):

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

where $\|\cdot\|$ is the matrix Frobenius norm. Some authors [61] use slightly different, but equivalent reformulation of it, namely:

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

After some transformations, the problem (1.2) can be reformulated as (see [9], Appendix A):

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

where $\otimes$ denotes the Kronecker product and $vec(\hat{P})$ 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 [9, 51].

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 $s_E : E^I \times E^J \to \mathbb{R}$ and should penalize disagreements in the structure of two graphs. The second one $s_V : V^I \times V^J \to \mathbb{R}$ 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:

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

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 $s_E(e_{ii\prime}, e_{jj\prime})$. Using this matrix we become the following formulation of the graph matching problem as an quadratic optimization problem [11, 13, 14, 18, 36]:

$$\operatorname*{argmax}_x x^T S x \tag{1.6}$$

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

$$\sum_{i=1...n_1} x_{ij} \leq 1 \tag{1.8}$$

$$\sum_{j=1...n_2} x_{ij} \leq 1 \tag{1.9}$$

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:

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

  It can be, for example, an euclidean distance ([13]cite). The other widely used alternative to define node similarity, is to use cosine similarity[2] of node attributes [43]cite: $dist_c(d_i, d_j) = \frac{d_i \cdot d_j}{\|d_i\|_2 \|d_j\|_2}$, where dot denotes scalar product of two vectors.

- Edge similarity

  One possible approach to calculate edge similarity is to calculate *edge dissimilarity* $d_E : E^I \times E^J \to \mathbb{R}$ first and then transform it into similarity by using, for example, one of the following functions [11, 12, 13, 14]:

  - $s_E(e_{ii\prime}, e_{jj\prime}) = exp(-\frac{d_(e_{ii\prime}, e_{jj\prime})^2}{\sigma_s^2})$
  - $s_E(e_{ii\prime}, e_{jj\prime}) = \max(\beta - d_(e_{ii\prime}, e_{jj\prime})/\sigma_s^2, 0)$

  where $e_{ii\prime} \in E^I$ and $e_{jj\prime} \in E^J$. 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 [11, 12, 13, 14], if coordinates

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

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 dissimilarity [12, 14]:

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

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

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

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* [7]. 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* [18].

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 [5]:

- 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 $V^I \backslash \bar{V}^I$ are *deleted*
- nodes in $V^J \backslash \bar{V}^J$ 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 $S = \{s_1, s_2, \ldots, s_k\}$ 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 [7, 62]:

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

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 [7]. The same author in [8] 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 [6] 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 $\sum_{i=1}^{n}\sum_{j=1}^{n}(A^I_{ii'} - A^J_{\sigma(i)\sigma(i')})^2$. 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:

$$c(e_{\text{subst}}) = (A^I_{ii'} - A^J_{\sigma(i)\sigma(i')})^2 \quad c(e_{\text{insert}}) = (A^J_{\sigma(i)\sigma(i')})^2 \quad c(e_{\text{del}}) = (A^I_{ii'})^2$$

1.5 Methods for solving inexact graph matching problems

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.

1.5.1 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 [28]. The first algorithm for inexact graph matching based on the three search was presented in [58]. It is an optimal inexact graph matching algorithm. For further examples of the three search methods for inexact graph matching we refer to [7, 54, 62].

Simulated Annealing

Simulated annealing is a heuristic for searching the global optimum of a given energy function [9]. It is an extension of the Metropolis Algorithm [42], 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 [29]. It is also a common used heuristic used for quadratic assignment problem [9].

1.5.2 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) [16]. 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 determenistic annealing was also used in [46] to solver Lagrangian Relaxation of the problem (1.2) and in the robust point matching algorithm with thin-plane spline (RPM-TPS) [17]. 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 [36] 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* [25] 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 Tailor series approximation to the objective function. The minimization is performed in a discrete using the Hungarian algorithm [32]. Then the initial objective function is further maximized in a continuous domain along found direction. The authors show, that in the praxis IPFP tens to converge towards discrete solutions, that are close to the optimum.

The Fast Approximative Quadratic Programming Algorithm (FAQ), presented in [61], 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

[3]Deterministic annealing is similar to the simulated annealing, but uses determenistic techniques to find the closest local minimum by value of the temperature parameter [48].

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 [64]. The algorithms finds first the global optimum of the Lagrangian relaxation of the problem (1.3). For this the Newton method [3] is used. Afterwards the found solution is projected back onto discrete domain by solving a sequence of convex and conclave problems. Those problems are solved again with Frank-Wolfe algorithm.

Very close to [36] is also the fast projected fixed-point algorithm (FastPFP) proposed in [37], 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 two last mentioned methods, because it does not need to solve linear assignment problem on each iteration.

Besides the mentioned methods, we want wo mention a work by Schellewald and Schnoerr, who use *a semidefinite programming* to solve graph matching problem. They formulate graph matching problem as a regularized bipartite matching problem with one continuous parameter. Then they relax this formulation to a convex semidefinite program. The obtained optimization problem is convex and can be solved with standard algorithms. The authors also suggest a novel probabilistic post-processing step to obtain a discrete solution from the continuous one. The suggested algorithm has only one tuning parameter, which makes it easy to apply. However the considered relaxation problem has a squared problem size comparing to initial problem [19].

Spectral methods

A big group of algorithm for solving graph matching problem is represented by those, which use a eigenvalue decomposition [3] 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 [60]. He considers the case, when a two graphs have the same number of nodes and match-

ing 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 resent algorithm is this group of methods is called spectral matching (SM) [35]. 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 weight[4]. This problem can be formulated as an integer optimization problem. However the authors relax both mapping and integer constrains and obtain the optimal 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 [22]. 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 [35]: 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.

Before going to the next section we want shortly describe a max-pooling approach from [11], which uses ideas similar to both GAGM [16] and SM [35], but replace one of the sum in a updating step with maximum-operation. The algorithm solves the graph matching problem formulation (1.6) by relaxing its integer constraints and omitting sum constraints. To approximate a solution of $\text{argmax}_x\, x^T S x$ the max-pooling algorithm similar to GAGM uses first order Taylor expansion of the objective function. The maximization of the Taylor approximation can be done by the power method as it is done in SM. The resulting update formula of the

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

correspondence $(v_i, v_j), v_i \in V^I, v_j \in V^J$ has the form [11]:

$$(Ax)_{ij} = x_{ij} s_V(v_i, v_j) + \sum_{i' \in N_i} \sum_{j' \in N_j} x_{i'j'} s_E(e_{ii'}, e_{jj'}), \tag{1.15}$$

where N_i denotes a direct neighborhood of a nodes v_i, i.e. set of its adjacent nodes. The max-poling algorithm uses a maximum function instead of the second sum in the second term, which leads to

$$(Ax)_{ij} = x_{ij} s_V(v_i, v_j) + \sum_{i' \in N_i} \max_{j' \in N_j} x_{i'j'} s_E(e_{ii'}, e_{jj'}). \tag{1.16}$$

This simple idea helps to suppress a noisy votes of the outlier matches, because the last have more likely low similarity values. The proposed algorithm shows very good results in presents of numerous outliers, although there is no theoretical justification of its convergence.

Probabilistic frameworks

Another group of algorithms applies *relaxation labeling* to solve graph matching problem. The problem is formulated as an assignment a set of labels to a set of objects. The earlier works on this formulation are [24, 49]. In term of the graph matching, object are the nodes of the first graph, and labels correspond to the nodes of the second graph. For each object there is a vector, which defines the initial probabilities of the labels to be assign to the objects. Those probabilities are computed based on some informations known about the objects (in graph matching problem those are node and edge attributes). The initial probabilities are then modified in iterative manner, until algorithm converges or riches the maximum number of iterations. The most advantage of this approach is polynomial complexity of the new problem formulation [15].

A first algorithm with theoretically proofed update rule is described in [30]. This algorithm however uses only node attributes and only to initialize probabilities, but not to update them. This two drawbacks were covered later by the work [15]. They use an update rule based on the Maximum A Posteriori probability (MAP) estimation.

Also in [63] graph matching problem is formulated as MAP estimation. The matching process here is a process of node deletion and reinstating, so that MAP rate monotonically increases in each iteration. This technique is suggested to effectively cope with the possible outliers in the sets of graph nodes. A node is considered as

an outlier, if its deletion improves the consistency in the structure of two graphs. The structural constrains hereby are defined by a dictionary of feasible mappings between local neighborhoods (super-cliques) of a nodes of two graphs. The algorithm shows good results in case of big number of outliers, is however slow.

Another interesting idea is presented in [38]. Given two graph, a nodes of the first graph are considered as an observed data and the nodes of the other as a hidden random variables. In this formulation the graph matching problem is solved using *Expectation-Maximization algorithm* (EM) [20]. A similar algorithm is presented in the resent paper [52]. However, it works with the continuous correspondences between the graph nodes and projected them in the discrete domain using soft-assign technique described previously. It also includes structural information[5] into matching process, whereby the algorithm from [38] uses only geometrical information. Both algorithm are however not applicable to a big graphs due to unlimited increasing of the density function with the size of a graphs. A new scalable graph matching algorithm, which also uses EM to solve the problem formulated in a maximum likelihood estimation framework, is presented recently in [2]. The scalability is achieved there by including information about already known correspondence into calculation of the density function.

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 [12] 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

[5] Structural relations between nodes are given through adjacency matrix of a graph. Geometrical relations are represented as relative position of the nodes with respect to each other [52].

in a small groups, so we can threshold them base on the cluster size.

Another algorithm based on the hierarchical clustering is described in [10]. 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 methods in the potential complexity reduction. This is achieved by performing graph matching not on the whole graphs, but on a smaller subgraphs, where the existing algorithms can find faster an approximate solutions. A solution of the whole problem is then obtained by combining local solutions. The great benefit of such approach is the ability to parallelize the computation process, which can lead to the significant time reduction. Qiu and Handcock suggest a clustering strategy based on a spectral method [45]. To build a partition of a graph into non-overlapping neighborhoods[6] they uses Fiedler vector (eigenvector associated with the second smallest eigenvalue of a graph Laplacian matrix, see [23]). Afterwards they show how to use estimated partition for graph matching (approach similar to one from [10] described above). They also provide a study, which shows, that the matching, that use proposed partition technique, is stable under structural errors. Additionally, the proposed clustering algorithm can be used to create hierarchical simplification of an initial graph, where the cluster represent nodes of a simplified graph.

The same idea of using graph clustering is followed by Lyzinski et al. in their paper [39]. The aim of the authors is to parallelize a graph matching algorithm for matching very big graphs. However the problem, they consider, 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.

[6] A node neighborhood is defined here as a node with all its adjacent nodes.

Resent algorithms for big graphs

1.6 Graph matching algorithms studied in this thesis

In this section we want for completeness described detailed graph matching algorithm used in our framework for graph matching. However, we do not provide theoretical justification of the algorithm steps and refer a reader to the original paper for that. We use the notation introduced at the beginning of this chapter.

1.6.1 Reweighted random walks for graph matching (RRWM)

The presented here algorithm is designed by Minsu Cho et al. [13] and interprets the graph matching problem (1.6):

$$\underset{x}{\operatorname{argmax}}\, x^T S x \tag{1.6}$$

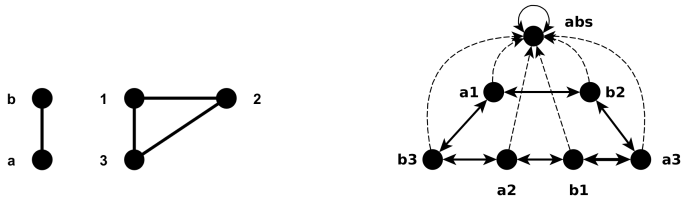
$$\text{s.t. } x \in \{0,1\}^{n_1 n_2} \tag{1.7}$$

$$\sum_{i=1\ldots n_1} x_{ij} \leq 1 \tag{1.8}$$

$$\sum_{j=1\ldots n_2} x_{ij} \leq 1 \tag{1.9}$$

between two graphs $G^I = (V^I, E^I, D^I)$, $G^I = (V^I, E^I, D^I)$ in a random walk view. For this the authors define an association graph $G^{rw} = (V^{rw}, E^{rw}, D^{rw})$ based on the affinity matrix S. The set of nodes V^{rw} of the new graph is represented by all possible correspondences between nodes in V^I and V^J. This means, that $|V^{rw}| = n_1 n_2$, where $|V^I| = n_1$ and $|V^J| = n_2$. We refer to a node of the graph G^{rw} as v_{ij} if it represent a correspondence pair (v_i, v_j), $v_i \in V^I, v_j \in V^J$. The entry $S_{ij,i'j'}$ of the matrix S defines the weight of the edge $\{v_{ij}, v_{i'j'}\} \in E^{rw}$. We denote an weighted adjacency matrix of the graph G^{rm} as W. An example of the construction is illustrated on the figure below. Note, that we omitted contradiction edges, whose weights are equal zero, such as $\{a1, b1\}$.

It is obvious, that the graph matching problem between two graphs G^I and G^I is equivalent to finding a subset of nodes in the associated graph G^{rw}, so that the respective node correspondences satisfy matching criteria of the initial problem. For finding such a subset the authors adopt the page ranking algorithm based on a random work, which is assumed to be a Markov chain [44]. To describe a Markov chain one defines a transition matrix P. An usual approach to define a transaction matrix of a wighted graph G^{rw} is to convert the weighted adjacency matrix W of the



(a) Two graphs to match **(b)** Association graph

Figure 1.3: Association Graph of two given graphs for Reweighted Random Walk Method (compare to [13])

graph into a stochastic matrix by the following normalization $P = D^{-1}W$, where D is a diagonal matrix with entries $D_{kk} = \sum_l W_{kl}$. This method was, for example, used in PageRank algorithm [44]. It is however not suitable for the graph matching proposes, as it treats false correspondences equal to all other correspondences. To avoid this problem Cho et al. introduce an additional absorbing node v_{abs} (see Fig. 1.3b) in the Graph G^{rw}, which represent a state, that can be reached from each node $v_k = v_{ij} \in V^{rw}$ with the probability $1 - D_{kk}/D_{\max}, D_{\max} = max_k D_{kk}$, but can not be left any more. Using $D_{\max}$ the matrix W is converted into a stochastic matrix by its multiplication with the factor $1/D_{\max}$. Summarizing, the transition matrix $P \in \mathbb{R}^{(n_1 n_2+1)\times(n_1 n_2+1)}$ is defined as

$$P = \begin{pmatrix} W/D_{\max} & \mathbf{1} - d/D_{\max} \\ 0 \dots 0 & 1 \end{pmatrix} \tag{1.17}$$

and update formula of the probability distribution of the Markov chain as

$$\left(x^{(n+1)T},\ x_{\text{abs}}^{(n+1)}\right) = \alpha \left(x^{(n)T},\ x_{\text{abs}}^{(n)}\right) P + (1-\alpha) r^T, \tag{1.18}$$

where $\mathbf{1}$ in (1.17) denotes a vector of size $\mathbb{R}^{n_1 n_2 \times 1}$ with all entries equal to 1, $d = (D_{11}, \dots, D_{n_1 n_2})$ is the diagonal of the matrix D and $r \in \mathbb{R}^{n_1 n_2+1}$ is *reweighted jump vector*.

A Markov chain, defined by Eq. 1.18 without second term, is denoted as an *affinity-preserving random walk*. The distribution $\bar{\mathbf{x}}$ of unabsorbed random walks at time n is defined as follows:

$$\bar{\mathbf{x}}_{ij}^{(n)} = P(X^{(n)} = v_{ij} | X^{(n)} \neq v_{\text{abs}}) = \frac{x_{ij}^{(n)}}{1 - x_{\text{abs}}^{(n)}}, \tag{1.19}$$

where $X^{(n)}$ is a current location of a random walker at time n. The authors call $\bar{\mathbf{x}}$ a *quasi-stationary distribution* of the absorbed Markov chain.They proof, that the distribution $\bar{\mathbf{x}}$ is proportional to the left principal eigenvector of W and can be efficiently computed with power iteration method [27].

The second summand in the Eq. 1.18 represents the possibility of a random walker to make a jump with probability $(1-\alpha)$ into some constrained node, instead of following the edge. This term was proposed by the authors based on personalization approach for web pages ranking [33] as a way to include the matching constraints (1.8), (1.9) into random walk. The authors are pointing out, that without this term the matching constrains are incorporated only in the last discretization step of the algorithm, which leads to a weak local maximum.

A procedure of generation the jump vector r from a current quasi-stationary distribution $\bar{\mathbf{x}}$ consists of two steps. In the first step (*inflation*) unreliable correspondences (i.e. small values in the vector $\bar{\mathbf{x}}$) are damped and the good correspondences are at the same time boosted. The second step (*bistochastic normalization*) forces the matching constraints by transforming the matrix form of $\bar{\mathbf{x}}$ into double stochastic matrix using the normalization scheme of Sinkhorn [57].

The described steps are summarized in Algorithm 1 below:

Algorithm 1: Reweighted Random Walks Method, compare to [13]

Input: weight matrix W, the reweight factor α, the inflation factor β

Output: distribution $\mathbf{x}$

1 set $W_{ij,i'j'}$=0 for all conflicting match pairs, i.e. $(v_{ij}, v_{ij'})$ and $(v_{ij}, v_{i'j})$

2 $D_{\max} = \max_{ij} \sum_{i'j'} W_{ij,i'j'}$

3 $P = W/D_{\max}$, initialize starting probability $\mathbf{x}$ as uniform

4 **repeat**

`/* Affinity preserving random walking by edges */`

6 $\bar{\mathbf{x}} = \mathbf{x}^T P$

`/* Reweighting with two-way constraints */`

`/* step 1 inflation: */`

7 $\mathbf{y}^T = \exp(\beta \bar{\mathbf{x}} / \max \bar{\mathbf{x}})$

`/* step 2 bistochastic normalization : */`

8 **repeat**

9 normalize across rows by $\mathbf{y}_{ij} = \mathbf{y}_{ij} / \sum_j \mathbf{y}_{ij}$

10 normalize across columns by $\mathbf{y}_{ij} = \mathbf{y}_{ij} / \sum_i \mathbf{y}_{ij}$

11 **until** $\mathbf{y}$ *converges* ;

12 $\mathbf{y} = \mathbf{y} / \sum \mathbf{y}_{ij}$

`/* Affinity-preserving random walking with reweighted jumps */`

13 $\mathbf{x}^T = \alpha \bar{\mathbf{x}}^T + (1 - \alpha)\mathbf{y}^T$

14 $\mathbf{x} = \mathbf{x} / \sum \mathbf{x}_{ij}$

15 **until** $\mathbf{x}$ *converges* ;

16 discretize $\mathbf{x}$ by the matching constraints

The discretization step in the line 16 1 can be done by using any method, which solves the linear assignment problem, i.e. Hungarian algorithm [32] or greedy heuristic as in [35].

The complexity of the algorithm is $\mathcal{O}(|E^I||E^J|)$ per iteration, whereby the quasi-stationary distribution $\bar{\mathbf{x}}$ in line 6 1 was computed with the power iteration method [27].

There are some similarities between RRWM and some algorithms, described in the chapter 1. For example, the authors notice, that the line 6 1 of the Algorithm 1 can be considered as the power iteration version of SM [35]. Also the Sinkhorn normalization (line 8-11 1) was used in soft-assign step in [16].

In the chapter ref we discuss experimental results of the RRWM for graph matching.

1.7 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 [9, 31]:

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

where Σ_n is a set of all possible permutations of the set $\{1,\ldots,n\}$, 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

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

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

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

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.

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

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

From the definition of the matrix trace follows cite:

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

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

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

We can now apply the equality 2 and 3 in (A.4) to the formulation (A.2) and get the *trace formulation of QAP* [9]:

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

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

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

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):

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

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

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] A. Armiti and M. Gertz. Geometric graph matching and similarity: A probabilistic approach. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, SSDBM '14, pages 27:1–27:12, New York, NY, USA, 2014. ACM.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[4] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[5] 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.

[6] 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.

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

[8] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259, 1998.

[9] 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 Combi-*

natorial Optimization, pages 241–338. Kluwer Academic Publisher, 1998.

[10] M. Carcassoni and E. R. Hancock. Correspondence matching with modal clusters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(12):1609–1615, 2003.

[11] 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.

[12] 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.

[13] M. Cho, J. Lee, and K. M. Lee. Reweighted Random Walks for Graph Matching. *ECCV*, 2010.

[14] 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.

[15] W. J. Christmas, J. Kittler, and M. Petrou. Structural Matching in Computer Vision Using Probabilistic Relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):749–764, 1995.

[16] 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.

[17] H. Chui and A. Rangarajan. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, 89:114–141, 2003.

[18] 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.

[19] T. Cour, P. Srinivasan, and J. Shi. Balanced graph matching. In *Advances in Neural Information Processing Systems (NIPS)*, pages 313–320, 2006.

[20] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[21] R. Diestel. *Graph Theory, Electronic Edition 2005*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag New York, 2005.

[22] 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.

[23] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(4):619—-633, 1975.

[24] M. Fischler and R. Elschlager. The Representation and Matching of Pictorial Structures. *IEEE Transactions on Computers*, 22(1):67–92, 1973.

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

[26] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[27] G. Golub and C. Van Loan. *Matrix Computations*. Last access on 28/09/2015.

[28] P. E. Hart, N. J. Nillson, 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.

[29] L. H'erault, R. Horaud, F. Veillon, and J. Niez. Symbolic image matching by simulated annealing. In *Procedings of the British Machine Vision Conference*, pages 319–324, 1990.

[30] J. Kittler and E. R. Hancock. International journal of pattern recognition and artificial intelligence. *IEEE Trans. Pattern Anal. Mach. Intell.*, 3(1):29–51, 1989.

[31] 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.

[32] H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterlyg*, 2:83–97, 1955.

[33] A. Langville and C. Meyer. Deeper Inside PageRank. *Internet Mathematics*, 1(3):335–380, 2003.

[34] 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.

[35] M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. In *ICCV*, 2005.

[36] M. Leordeanu, M. Hebert, R. Sukthankar, and M. Herbert. An Integer Projected Fixed Point Method for Graph Matching and MAP Inference. In *NIPS*, 2009.

[37] Y. Lu, K. Huang, and C.-L. Liu. A fast projected fixed-point algorithm for large graph matching, 2012.

[38] B. Luo and E. R. Hancock. Structural graph matching using the EM algorithm and singular value decomposition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(10):1120–1136, 2001.

[39] 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.

[40] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *CoRR*, 2013.

[41] B. Messmer and H. Bunke. A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recognition*, 32(12):1979–1998, 1999.

[42] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chenical Physics*, 21:1087–1092, 1953.

[43] 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.

[44] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[45] H. Qiu and E. R. Hancock. Graph matching and clustering using spectral partitions. *Pattern Recognition*, 39(1):22–34, 2006.

[46] A. Rangarajan and E. Mjolsness. A lagrangian relaxation network for graph matching. In *IEEE Trans. Neural Networks*, pages 4629–4634. IEEE Press, 1996.

[47] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory*

and Practice. Prentice Hall College Div, 1977.

[48] K. Rose. Determenistic annealing, clustering and optimization, 1991.

[49] A. Rosenfeld, R. a. Hummel, and S. W. Zucker. Scene Labeling by Relaxation Operations. *IEEE Transactions on Systems, Man, and Cybernetics*, 6(6):420–433, 1976.

[50] 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.

[51] S. Sahni. Computationally Related Problems. *SIAM J. Comput*, 3(4):262–279, 1974.

[52] G. Sanromà, R. Alquézar, and F. Serratosa. A new graph matching method for point-set correspondence using the EM algorithm and Softassign. *Computer Vision and Image Understanding*, 116:292–304, 2012.

[53] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.

[54] 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.

[55] K. Shearer, H. Bunke, and S. Venkatesh. Video sequence matching via decision tree path following. *Pattern recognition letters*, pages 479–492, 2001.

[56] 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.

[57] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *Annals of Mathematical Statistics*, 35(2):876—-879, 1964.

[58] 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.

[59] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[60] S. Umeyam. An Eigendecomposition Approach to W e ighted G raph Matching Problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*,

10, 1988.

[61] 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.

[62] J. T. Wang, K. Zhang, and G.-W. Chirn. Algorithms for approximate graph matching. *Information Sciences*, 82(1-2):45–74, 1995.

[63] R. C. Wilson, A. D. J. Cross, and E. R. Hancock. Structural matching with active triangulations. *Computer Vision and Image Understanding*, 72(1):21–38, 1998.

[64] 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) .....