

Contents

1	Graph Matching	2
1.1	Basic definitions and notations	2
1.2	Exact graph matching	4
1.3	Methods for solving exact graph matching problems	5
1.4	Inexact graph matching	6
1.4.1	Graph matching objective function	7
1.5	Methods for solving inexact graph matching problems	12
1.5.1	Discrete optimization	12
1.5.2	Continuous optimization	13
1.6	Graph matching algorithms studied in this thesis	20
1.6.1	Reweighted random walks for graph matching (RRWM)	20
1.7	Discussion	23
2	Two level graph matching	24
2.1	Problem statement	25
2.2	Algorithm	27
2.2.1	Anchor Graph Construction	30
2.2.2	Anchor graph matching	31
2.2.3	Subgraph matching	33
2.2.4	Graph matching algorithm	34
2.2.5	Graph partition update	34
2.3	Discussion	37
A	Quadratic Assignment Problem	38
B	Bibliography	41

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 recent 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 $E \subseteq \{\{u, v\} | u, v \in V\}$ [22]. 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* $A = (a_{ii'})_{n \times n}$, where

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

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 $i = 1, \dots, k$. 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. A subgraph G' of G is *induced by a node subset* $V' \subseteq V$, 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 $G' = G[E']$ [22], if it is node- or vertex-induced respectively. We also introduce a 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* $G = (V, E, D)$. In contrast to this, if 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*).

Let us 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 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 $m : V^I \rightarrow V^J$, 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 [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.

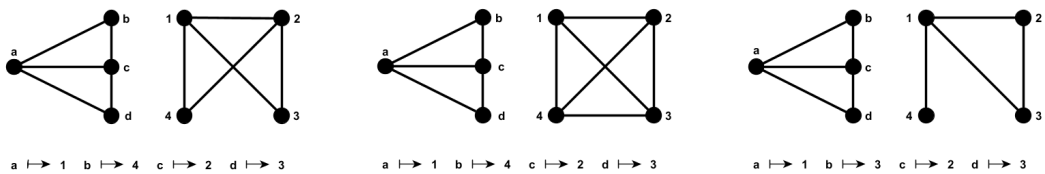
1.2 Exact graph matching

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 *isomorphic* ($G^I \simeq G^J$), 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 (i.e. every node $v_i \in V^I$ has to be mapped into V^J). On the Fig. 1.1 one can see examples of different exact graph matching problems. Node correspondences are listed below each subfigure.



(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 [27]. This can be shown through a reduction of the respective matching problem to the clique problem. The graph isomorphism is currently shown to be in the class NP [27, 61]. For some special types of graphs there exist however polynomial time algorithms (e.g. for trees [1, 27]).

The exact graph matching problems are often too strict and intractable for the practical application and therefor do not lie in the focus of this work. In following we describe several approaches for solving them.

1.3 Methods for solving exact graph matching problems

The most used approach to solve an exact graph matching problem is based on a tree search with backtracking. This method starts with an empty set of correspondences and tries stepwise to expend it according to provided rules until a complete solution is found. Each partial solution represent a node of a search tree and all nested solutions build a branch in this tree. If it happens, that in some step a current set of correspondences cannot be expended further due to problem constraints, the current branch in the tree is cut. The method backtracks to the last feasible solution and tries to find another way to expend it further. The algorithms based on a tree search are very slow, if they need to traverse a whole tree. However, they can be speeded up by applying some heuristics 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 algorithm [53].

The one of the first algorithms, that used the described technique, is the one by Ullman [68]. Later it was extended and improved mostly by the suggestion of a new pruning heuristics. A small comparison of different algorithms in this group with diverse heuristics is reported in [38]. We also want to mention here another well known algorithm on graphs, which uses the tree search method, namely the algorithm by Bron and Kerbosch [4] for finding cliques in an undirected graphs. The last problem closely related to the graph matching, as the maximal common subgraph problem can be reduced to the problem of finding the maximal clique cite.

From the other techniques we want to mention the algorithm described by MacKay [45], which uses group theory to solve graph isomorphism problem, and an approach based

on decision trees [46, 63, 64] for matching graph ((sub)graph isomorphism) against a set of graphs.

1.4 Inexact graph matching

As we mentioned above exact graph matching problems are often not applicable to real-world problems. There are two possible reasons for that. **On the one hand, there are natural variations in the structure of graphs, that describe a same object** On the one hand, same object can be described by graphs with different structure. Those variations could be a consequence of object deformations or noise influence, that can occur in some real world applications. On the other hand, solving a graph matching problem exactly can be time and/or memory consuming. As a consequence one can be interested in solving graph matching problem inexactly. In this case, a strong edge preserving mapping between the nodes of two graphs is not required.

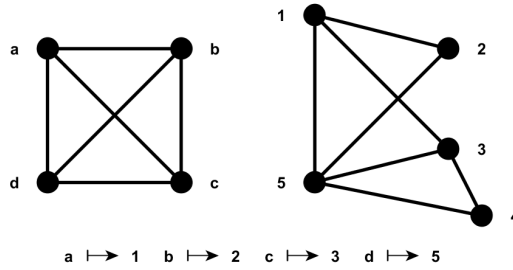


Figure 1.2: Inexact graph matching

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 a 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 on the case, if an algorithm for solving problem (1.1) finds a global solution or not, this algorithm is called *optimal* or *suboptimal*, respectively. The choice, which algorithm to select depends on a specific problem. It should be noted, that 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 forms of the objective function of the problem (1.1).

1.4.1 Graph matching objective function

In some literature instead of defining a similarity function as in (1.1) one speaks about dissimilarity between two graphs [32, 42, 44, 56, 71, 75]. The goal in this case is to minimize it. Two formulations of the problem are however equivalent and one can easily transform one into other.

Quadratic Optimization Problem

Let consider two attributed graphs $G^I = (V^I, E^I, D^I)$ and $G^J = (V^J, E^J, D^J)$. We want to find a mapping m between the nodes of this graphs. 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 a 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 first with a even stricter assumption, that $n_1 = n_2 = n$ and the matching is bijective. In this case a mapping between nodes of 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 [32, 42, 44, 56, 69, 75]):

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

where $\|\cdot\|$ is the matrix Frobenius norm. Some authors (e.g. Vogelstein et al. in [71]) use slightly different, but equivalent reformulation of it, namely:

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

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

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

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 known to be *NP*-hard [9, 58].

The both formulations (1.2), (1.4) 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 considers 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 a disagreements in the structure of two graphs. The second one $s_V : V^I \times V^J \rightarrow \mathbb{R}$ represent additional constraints 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:

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

This formula can be also expressed in matrix form. We define an *affinity* or *similarity matrix* $S \in \mathbb{R}^{n_1 n_2 \times n_1 n_2}$, 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 [12, 14, 15, 18, 28, 39, 40]:

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

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

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

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

We notice, that in the case, where $n_1 = n_2$, both conditions (1.8) and (1.9) will be fulfilled with equality. We call this constraints *two-way constraints*, as they enforce one-to-one matching.

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.

Edge similarity

One possible approach to calculate an 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 [12, 13, 14, 15]:

$$\bullet \quad s_E(e_{ii'}, e_{jj'}) = \exp\left(-\frac{d_E(e_{ii'}, e_{jj'})^2}{\sigma_s^2}\right) \quad (1.10)$$

$$\bullet \quad s_E(e_{ii'}, e_{jj'}) = \max(\beta - d_E(e_{ii'}, e_{jj'})/\sigma_s^2, 0) \quad (1.11)$$

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 [12, 13, 14, 15], if coordinates of the nodes in some system (usually, Cartesian coordinates) are known. It is a significant assumption, which holds however for almost all graphs arose in practical application.

Sometime a node attributes can be used to calculate edge dissimilarities. For example, if graph nodes are described by an ellipse with known center coordinates and orientation one can calculate so-called geometric dissimilarity of the edges [13, 15]:

$$d(e_{ij}, e_{i'j'}) = \frac{1}{2}(d_{geo}(m_j|m_i) + d_{geo}(m_i|m_j)) \quad (1.12)$$

$$d_{geo}(m_j|m_i) = \frac{1}{2}(\|x_{j'} - H_i x_j\| + \|x_j - H_i^{-1} x_{j'}\|) \quad (1.13)$$

$$d_{geo}(m_i|m_j) = \frac{1}{2}(\|x_{i'} - H_j x_i\| + \|x_i - H_j^{-1} x_{i'}\|) \quad (1.14)$$

where m_i is a correspondence between nodes v_i and $v_{i'}$ and H_i is an affine homography from v_i to $v_{i'}$ estimated based on elliptic regions around each node.

Another way to calculate edge similarities could be to use directly some similarity measure, such as cosine similarity.

Node similarity

If a given graphs have node attributes, the most direct method to measure a similarity of two nodes is to compare their attributes. In the most of the seen literature, the node attributes are represented by a k -dimensional real vectors: $D^i, D^j \in \mathbb{R}^k$. This means, we can adopt all techniques² described in previous paragraph to define a node similarity function. Additionally, in case when node attributes can be considered as some distribution (e.g. distribution of gray values of an image around a node), one can think to use metrics to measure distance between two distributions. For example, Schellewald and Schnörr [60] used the earth mover distance for this propose.

Error correcting graph matching

Another way to measure the similarity between two graphs is based on a *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 nodes and edges of a graph. An algorithm, that uses graph edit distance for graph matching, is 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]:

²with except of the geometric dissimilarity measure

- 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 a transformations applied to its end nodes. For example, insertion of a node $v_j \in V^J$ implies that all edges, connected to this node, are also inserted. Alternatively, if a node $v_i \in V^I$ is 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 we define a *edit distance* between the graphs G^I and G^J as follows [7, 72]:

$$\text{dist}(G^I, G^J, m) = \min_{S=\{s_1, \dots, s_k\}} c(S) \quad (1.15)$$

The task of the error-corrected graph matching is to find such a mapping m between V^I and V^J , which induces a cheapest transformation between the graphs.

The cost functions of each operation are often defined as a functions of edge weights and node attributes. Their exact definitions depend however on a specific application. Sometimes, it can be helpful, when the distance measure (1.15) defines a metric, which is not automatically the case. Bunke and Allermann [7] have shown that, the graph edit distance can fulfill the properties of a metric under a specific conditions on the cost functions of the individual edit operations. For example, one if this conditions is, that a single operation is always preferred to a sequence of two operations with the same result (triangular inequality of the operation cost functions). Later, Bunke and Shearer [8] suggest a new graph edit distance, that is not depending on the cost of the edit operations, and is a metric.

An interesting question is, how does the error corrected graph matching problem correspond to the other matching problems defined previously. It was shown in [6], that graph isomorphism, subgraph isomorphism and maximum common subgraph problems can be considered as a special cases of the error correcting graph matching. The same holds for the inexact graph matching problem formulated as in (1.2). This is easy to see, if one rewrite the objective function of

(1.2) as $\sum_{i=1}^n \sum_{j=1}^n (A_{ii'}^I - A_{\sigma(i)\sigma(i')}^J)^2$. Comparison of this expression with the definition of $c(S)$ in (1.15) leads us direct to the idea, how to set the edge insertion/deletion/substitution cost functions, to make the problem formulations equivalent:

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

1.5 Methods for solving inexact graph matching problems

In following we briefly describe a common approaches for solving inexact graph matching problems 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 a tree based methods with backtracking were successfully applied to solve inexact matching problems. They were especially wild used for the error-correcting graph matching. In this case a searching procedure can be efficiently guided, so that the required time is shorter than exponential time, which is needed by a blind searching. This can be done by defining the score of the tree nodes as a sum of two scores: a matching score of a current partial solution and a heuristically estimated score of a remaining nodes. The algorithms in this group differ mainly in a suggested heuristics for calculation of future costs and in a rules, for tree traversing. The most used strategies for tree traversing are depth-first-search [19] and A^* -Algorithm [30].

One of the first algorithms for inexact graph matching based on the three search was presented by Tsai and Fu in [67]. It is an optimal inexact graph matching algorithm. For further examples of the three search methods for inexact graph matching we refer to [7, 62, 72].

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 [47], that was

suggested for finding an equilibrium of a physical system consisting of individual particles by heating it first and then cooling it slowly down. Speaking in terms of a graph matching problem, its solutions define states of a system and corresponding objective scores its energy. The idea is to find a state with the lowest energy by performing some random changes in a system. This can be a random exchange of correspondences between two pairs of matched nodes. A change, that increases the total dissimilarity of graphs, 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 [32]. This heuristic can also be used to solve a quadratic assignment problem [9].

1.5.2 Continuous optimization

This group of methods is one of the biggest and deep investigated. The main idea is to transform the inexact graph matching problem into a continuous optimization problem. This can be done, for example by relaxing the integer constraints in the discrete optimization problems (1.2), (1.6). A found solution must be converted afterwards back in the discrete domain. Depending on techniques, applied to solve a continuous problem, we subdivide this group into several subgroups. The list of the algorithms, presented here, is not complete. As we already mentioned, we selected mainly the classical approaches and/or those, we investigated during the work on this thesis.

The first subgroup of the algorithms contains the algorithms, that relax all constraints of a graph matching problem, find a continuous solution of the new nonlinear problem and project it back into discrete domain.

In 1996 Gold and Rangarajan presented a graduated assignment algorithm for graph matching (GAGM) [28]. It is an iterative algorithm, which uses *deterministic annealing*³ to solve the graph matching problem formulated as (1.6). The authors use Taylor series approximation to relax the initial quadratic assignment problem to linear assignment problem. The two-way constraints are enforced at each itera-

³Deterministic annealing is similar to the simulated annealing, but uses deterministic techniques to find a minimum of an energy function by a current value of a temperature parameter [54].

tion by converting a found continuous solution into a double stochastic matrix by applying exponentiation and Sinkhorn normalization [65]. This technique is called *soft-assign*. The deterministic annealing was also used in [52] to solve 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 necessarily 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 [40] 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* [26] adapted to the graph matching problem. This algorithm finds first the best search direction by solving a linear assignment problem. The last arises as in GAGM by the maximization the first-order Taylor series approximation of the objective function. The maximization is performed in the discrete domain using the Hungarian algorithm [36]. 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 [71], 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. Unlike IPFP, the FAQ algorithm do not use the similarity matrix S . This gives it a big advantage of the less space requirement, which make it possibly to apply the algorithm to graphs of a big size⁴.

Another approach, that uses Frank-Wolfe algorithm, is the PATH-Algorithm [74]. The algorithm 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, which are solved Frank-Wolfe algorithm.

Very close to IPFP is the fast projected fixed-point algorithm (FastPFP) proposed by Lu et al. [42], that uses *projected fix point method* to find a solution of a contin-

⁴Evaluation results in the paper include test with a graphs with up to 1000 nodes

uous 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 it does not use a similarity matrix S and is generally faster than both IPFP and FAQ, because it does not solve a linear assignment problem on each iteration.

A completely different approach to solve graph matching problem was suggested by Schellewald and Schnörr [60]. They formulated graph matching problem as a regularized bipartite matching problem [22], which is then relaxed to a convex semidefinite program. The last can be solved with standard algorithms, such as the interior point method [3]. A novelty of the algorithm consists in a probabilistic post-processing step, which convert a found continuous solution into discrete one. The suggested algorithm is easy to apply, as it has only one parameter. However the considered relaxation problem has a squared size comparing to initial problem [20]. Consequently, the algorithm is not suitable for big graphs.

Spectral methods

A big group of algorithms for solving graph matching problems is built by those, which use an eigenvalue decomposition [3] of the adjacency matrices of given graphs. The idea behind this is, that the eigenvalues of the adjacency matrices of two isomorphic graphs are equal, because they are invariant to the node permutation⁵.

One of the first algorithms, which uses spectral techniques, is the one by Umeyama [69]. He considers the case, when two graphs have the same number of nodes and matching between them is bijective. This means, that a desired correspondence matrix must be a permutation matrix. The author proofs, that an orthogonal matrix, which minimizes the objective function of (1.2), can be formulated based on the eigenvalue decomposition of the adjacency matrices of given graphs. The terms of this formulation are used to define a linear assignment problem, whose solution is a solution of the initial problem. In case, when given graphs are sufficiently close, those solution is optimal or nearly optimal.

We notice, that this algorithm is highly limited in its application due to its require-

⁵Indeed, let A and B be adjacency matrices of two isomorphic graphs. That means, that $B = PAP^T$, where P is a permutation matrix. If α is an eigenvalue of A : $Av = \alpha v$ and $u = Pv$, then the following holds $Bu = (PAP^T)u = (PA)(P^T u) = (PA)v = P(\alpha v) = \alpha u$. So α is an eigenvalue of B

ments. We also want to mention, that it works only with graph geometry and does not consider attributes of nodes and edges.

The more recent algorithm in this group of methods is called spectral matching (SM) [39]. The algorithm works with the general formulation (1.6). It relaxes both two-way and integer constraints (1.7)-(1.9) and obtains the optimal solution of this relaxation by taking the principal eigenvector of the matrix S . This is done by using the *power iteration method* [29]. The found continuous solution is binarized by applying a simple greedy heuristic, that enforces previously relaxed matching constraints.

A generalization of the SM algorithm is proposed in [23]. The authors suggest to consider similarities between tuples of nodes and not only pairs to improve graph matching quality. They formulate a new problem using hyper-graphs and tensor notation. The solution strategy is however the same, as in [39]: the solution is approximated with the principal eigenvector of the matrix S and discretized afterwards. However, the power iteration method should be adopted to the tensor based problem, as it cannot be applied directly.

Before going to the next section we want to shortly describe a max-pooling approach by Cho and Duchenne [12], which uses ideas similar to both GAGM [28] and SM [39], but replaces one of the sum in an 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 $\arg\max_x x^T S x$ 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 correspondence $(v_i, v_j), v_i \in V^I, v_j \in V^J$ has the form:

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

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

$$(Ax)_{ij} = x_{ij} s_V(v_i, v_j) + \sum_{i' \in N_i} \max_{j' \in N_j} x_{i'j'} s_E(e_{ii'}, e_{jj'}). \quad (1.17)$$

This simple idea helps to suppress noisy entries 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 aim of the relaxation labeling is to find an optimal assignment between set of labels and set of objects. Each label has some initial probability to be assign to certain object. Those probabilities are set based on some informations known about the objects. In terms of graph matching problem, nodes of a one graph are considered as objects and nodes of another graph as labels. Initial probabilities of assignments between the nodes are calculated based on the node and edge attributes.

The earlier works on this formulation are the one by Fischer and Elschlager [25] and Rosenfeld et al. [55]. In both cases, proposed algorithms start with some initial label probabilities and try to reduce a labeling ambiguities at each iteration based on labels of neighboring nodes. The process runs till convergence or some predefined number of iterations. The algorithms are heuristic, but reduce the complexity of the initial problem to polynomial [16].

A first algorithm with theoretically proofed update rule is suggested by Kittler and Handcock [34]. This was later improved by Christmas et al. [16], who suggested a method to included provided node attributes and edge weights into update rule of the labeling probabilities and not only in the initialization step.

Later algorithms formulate graph matching problem as Maximum A Posteriori probability (MAP) estimation. For example the matching process in [73] is a process of node exclusion and insertion, 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 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 [43]. Given two graphs, nodes of the first graph are considered as an observed data and the nodes of the other as hidden random variables. In this formulation the graph matching problem is solved using *Expectation-Maximization algorithm* (EM) [21]. A similar algorithm is presented

in the recent paper [59]. However, it works with the continuous correspondences between the graph nodes and projects them in the discrete domain using soft-assign technique described previously. It also includes structural information⁶ into matching process, whereby the algorithm from [43] uses only geometrical information. Both algorithms are however not applicable to big graphs due to unlimited increasing of the formulated density function with the size of a graph [2]. 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 [13] proposes to use *agglomerative clustering* on a set of candidate matches to effectively cope with outliers. The problem is formulated 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 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 (1.12). Empirical results show, that correct correspondences tend to build bigger clusters. In contrast to that, the outliers are likely to stay in small groups, so it is possible to threshold them based on the cluster size.

Another algorithm based on the hierarchical clustering is suggested by Carcassoni and Hancock [10]. Its purpose 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 cluster centers. After performing the clustering, a probabilistic matching algorithm is applied to match first the clusters and then the nodes inside the clusters.

⁶Structural 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 [59].

Further benefit of using clustering methods in the potential complexity reduction. The first idea to achieve this is based on a hierarchical graph matching and appears for example in the work by Qui and Hancock [51]. They suggest to use graph partition to create a new simplified graph, whose nodes represent clusters of initial graph. After partitioning graphs are matched in a similar manner as it is suggested by Carcassoni and Hancock [10], where the derived clusters tie matching constraints. The used partition method is based on a spectral method and creates a set of non-overlapping node neighborhoods⁷. To build those they use Fiedler vector (eigenvector associated with the second smallest eigenvalue of a graph Laplacian matrix, see [24]). The authors provide a study, which shows, that the matching with proposed partition technique is stable under structural errors. Additionally, they show, that the simplified graphs preserve structure of initial graphs. The last was shown by performing clustering of a group of different graphs before and after simplification.

The second idea to reduce complexity of the initial matching problem uses graph clustering to create a set of independent subproblems. The matching is then performed for each subproblem independently and due to reduced problem size faster. 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.

The described idea is followed by Lyzinski et al. in their recent paper [44]. 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 the matching results of the subdivided problem depends highly on the quality of the graph clusters and established correspondences between the clusters of two graphs. In a perfect clustering, node that should be matched, must lie in the corresponding subgraphs. To ensure that the clustering is sufficiently good, the authors propose a technique, that first spectrally embeds graphs and then jointly cluster them. The algorithm was successfully applied to the graphs with 20000 – 30000 nodes.

⁷A node neighborhood is defined here as a node with all its adjacent nodes.

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. [14] and interprets the graph matching problem (1.6):

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

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

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

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

between two graphs $G^I = (V^I, E^I, D^I)$, $G^J = (V^J, E^J, D^J)$ 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 the weighted adjacency matrix of the graph G^{rw} as W . An example of the construction is illustrated on the figure below. Note, that we omitted contradiction edges, whose weights are equal zero, e.g. $\{a1, b1\}$.

It is obvious, that the graph matching problem between two graphs G^I and G^J is equivalent to finding a subset of nodes in the associated graph G^{rw} , so that the respective node correspondences between V^I and V^J 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 [50]. To describe a Markov chain one defines a transition matrix P . An usual approach

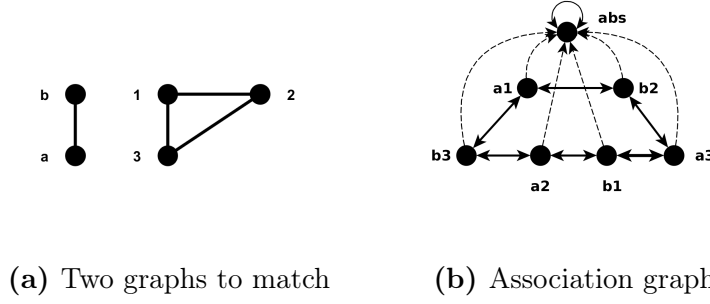


Figure 1.3: Association Graph of two given graphs for Reweighted Random Walk Method (compare to [14])

to define a transition matrix of a wighted graph G^{rw} is to convert the weighted adjacency matrix W of the 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 [50]. 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

$$P = \begin{pmatrix} W/D_{\max} & \mathbf{1} - d/D_{\max} \\ 0 \dots 0 & 1 \end{pmatrix} \quad (1.18)$$

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

$$\left(x^{(n+1)T}, x_{\text{abs}}^{(n+1)} \right) = \alpha \left(x^{(n)T}, x_{\text{abs}}^{(n)} \right) P + (1 - \alpha)r^T, \quad (1.19)$$

where $\mathbf{1}$ in (1.18) 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 , α is a weighted factor and $r \in \mathbb{R}^{n_1 n_2 + 1}$ is *reweighted jump vector*.

A Markov chain, defined by Eq. 1.19 without second term, is denoted as an *affinity-preserving random walk*. The distribution \bar{x} of unabsorbed random walks at time n

is defined as follows:

$$\bar{x}_{ij}^{(n)} = P(X^{(n)} = v_{ij} | X^{(n)} \neq v_{\text{abs}}) = \frac{x_{ij}^{(n)}}{1 - x_{\text{abs}}^{(n)}}, \quad (1.20)$$

where $X^{(n)}$ is a current location of a random walker at time n . The authors call \bar{x} a *quasi-stationary distribution* of the absorbed Markov chain. They proof, that the distribution \bar{x} is proportional to the left principal eigenvector of W and can be efficiently computed with power iteration method [29].

The second summand in the Eq. 1.19 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 [37] 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{x} consists of two steps. In the first step (*inflation*) unreliable correspondences (i.e. small values in the vector \bar{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{x} into double stochastic matrix using the normalization scheme of Sinkhorn [65].

The described steps are summarized in Algorithm 1 below:

The discretization step in the line 16 ?? can be done by using any method, which solves the linear assignment problem, i.e. Hungarian algorithm [36] or greedy heuristic as in [39].

The complexity of the algorithm is $\mathcal{O}(|E^I||E^J|)$ per iteration, whereby the quasi-stationary distribution \bar{x} in line 6 ?? was computed with the power iteration method [29].

There are some similarities between RRWM and some algorithms, described in the chapter 1. For example, the authors notice, that the line 6 ?? of the Algorithm 1 can be considered as the power iteration version of SM [39]. Also the Sinkhorn normalization (line 8-11 ??) was used in soft-assign step in [28].

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

Algorithm 1: Reweighted Random Walks Method, compare to [14]

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

```

1.7 Discussion

Unfortunately, most of the algorithms have the two following problems:

1. they are still limited in size of permissible graphs. Experiments in most of the papers consider graphs with up to 100 nodes [12, 14, 15]
2. possible presence of outliers can reduce the accuracy of matching algorithm [66].

Our main aim was to develop a framework, which would allow an existing graph matching algorithm to cope with both problems.

Chapter 2

Two level graph matching

In this chapter we describe our novel approach for graph matching. Our aim was not to develop a new matching algorithm, but to propose a framework, which would help to solve problems, where direct application of an existing matching algorithm is impossible due to memory requirements or performance time. This is often the case when a graph matching problem is formulated using a similarity matrix between graphs. Our main idea is based on the *divide and conquer* technique [19], which is well known from its application to array sorting algorithms. According to this general paradigm, a given graph matching problem, which is too difficult to be solved directly, is subdivided into smaller subproblems, those can be solved without great effort. A resulting solution is then combined from a local solutions of single subproblems. Important properties of such approach are its runtime improvement and not significant drop in accuracy. Desirable is off course also an improvement in the accuracy.

There are already several algorithms, which share the similar ideas. Those, which are most closed to our approach were described in previous chapter under algorithms based on clustering techniques. Below we review them shortly again to point out, that none of them completely repeat our framework.

This chapter is organized we follows. First of all, we formulate considered graph matching problem and show some issues of this formulation. In the second part, we describe our two level graph matching framework, which should help to cope with formulated problems. The performance results and comparison with other algorithms are summarized in the next chapter.

2.1 Problem statement

Consider two attributed graphs $G^I = (V^I, E^I, D^I)$ and $G^J = (V^J, E^J, D^J)$, where V, E, D denote set of nodes, set of edges and set of node attributes respectively. We assume the situation, where those graphs are undirected and do not have multiple edges between nodes. Let the size of the first graph be n_1 and the second n_2 . Without loss of generality, we assume that $n_1 \leq n_2$. Attributes of the graphs are k -dimensional real vectors: $D^I, D^J \in \mathbb{R}^k$.

We define a problem of matching two graphs G^I, G^J as a quadratic assignment problem (same formulation as (1.6)-(1.9)).

$$\operatorname{argmax}_x x^T S x \quad (2.1)$$

$$\text{s.t. } x \in \{0, 1\}^{n_1 n_2} \quad (2.2)$$

$$\sum_{i=1 \dots n_1} x_{ij} \leq 1 \quad (2.3)$$

$$\sum_{j=1 \dots n_2} x_{ij} \leq 1 \quad (2.4)$$

We denote a pair of nodes (v_i, v_j) , where $v_i \in V^I$ and $v_j \in V^J$, as a correspondence between the sets V^I and V^J . Let M be a set of all possible correspondences between the nodes of the graphs G^I, G^J . Obviously, M consists of $n_1 n_2$ node pairs. Then the vector $x \in \{0, 1\}^{n_1 n_2}$ is an indicator vector of a subset $m = \{(v_i, v_j) | v_i \in V^I, v_j \in V^J\}$ of the M . That means, that element x_k of this vector is equal 1 if and only if the corresponding k -th node pair (v_i, v_j) is selected into subset m . The constraints (2.3), (2.4) ensure, that each node of the graph G^I is matched to exactly one node of the second graph G^J .

The matrix $S \in \mathbb{R}^{n_1 n_2 \times n_1 n_2}$ in (2.1) encloses the precomputed information about similarity of two graphs. Rows (columns) of this matrix correspond to the elements in the set M of all possible node correspondences. Its diagonal element S_{kk} contains similarity measurement of the node pair $(v_i, v_j)_k \in M$. On the other side, the non-diagonal elements of S_{kl} measure similarity of edges between two pairs of matched nodes. Our aim is to find a subset of maximal n_1 correspondences between the nodes of the graphs G^I, G^J , which maximizes the similarity value between those graphs.

As we saw in the previous chapter, the selected formulation of a graph matching problem is widely used as the most general one. However, the size of the affinity matrix S can cause problems due to required memory demand. For example, a

dense affinity matrix between two graphs with 200 nodes each needs approximately 12Gb memory (double precision). There are several possible ways to reduce the memory complexity of the formulated graph matching problem. Here we mention three possible approaches.

The first one is to reduce a set of candidate correspondences by selecting a subset $M' \subset M$. This can be done, for example, by restricting a number of candidate matches for a node $v_i \in V^I$ to some number smaller than n_2 . This method is often used, as it not only solves memory issue of the problem formulation (2.1)-(2.4), but also reduces the algorithmic complexity of many algorithms, which highly depend on the number of possible matches (e.g. [12, 14, 15, 39]).

The second possibility, is to make the matrix S sparser by excluding comparison of some nodes or edges from consideration. In case of a big graphs this can however lead to a high loss of initially provided information and results dramatically on the quality of a resulting matching.

The third possibility is to replace an initial problem of graph matching by a set of smaller subproblems by partitioning given graphs into subgraphs and matching those subgraphs. For the matrix S it means, that it is divided into blocks, where each block represent a similarity matrix between two subgraphs. Thereby the similarities of edges, whose nodes belong after problem splitting to different subgraphs, will be ignored. On the one hand, this approach solves the memory problem by replacing the initial matrix S with a set of smaller affinity matrices. On the other hand, it does not reduce the algorithmic complexity of the initial problem, because the subgraphs of two graphs should be matched in all possible combinations between each other. Otherwise, further information will be lost. Despite the mentioned drawbacks, single subgraph matching problems can be eventually parallelized, what still makes the approach attractive for application on big graphs.

In the framework for graph matching, that we describe in the details below, we use the third of described techniques. We divide a given initial graphs into subgraphs and iteratively search first for correspondences between the subgraphs and then for node correspondences between matched subgraphs. For subgraph matching we use some existing matching algorithm. A graph partitioning is performed only at the initialization step, but after each iteration subgraphs have a change to exchange nodes on their border.

To our best knowledge the described method was not published before. Especially,

we haven't seen an iterative algorithm based on graph clustering so far, which would update initial partitions. At the same time there is a certain overlap in ideas between our and existing works. The algorithm proposed by Lyzinski et al. [44] uses graph partitioning to parallelize a semi-supervised graph matching problem, where some correspondences between graphs nodes are provided. The graph matching problem is formulated as the minimization problem (1.2), that does not use an affinity matrix S . The given matches between graph nodes are used to cluster two graphs jointly and to find a correspondences between subgraphs. As a consequence, subgraphs of given graphs are similar enough to ensure the matching quality. However the proposed clustering method cannot be used for a unsupervised matching.

The similar idea to our to use graph partition for graph matching in unsupervised case was used by Carcassoni and Hancock [10], Qui and Hancock [51] and recently by Nie et al. [49]. From them only the third algorithm consider the same maximization problem as we. The two other algorithms formulate graph matching problem in terms of relaxation labeling. Also the definition of the graph clusters differs between the algorithms. Qui and Hancock, as well as Nie et al., consider clusters, that are built by a direct neighborhoods of nodes. The resulting partition can be overlapping [49] or not [51]. Our algorithm and the one in [10] consider more general case, where a graph partition is given by a disjoint set of graph subgraphs. Finally, similar to our approach Qui and Hancock [51] use the extracted graph partitioning to create a new graph, whose nodes represent clusters of initial graphs. They call this process graph simplification. However their approach quite differs from our, because of special definition of clusters, another problem formulation and different approach for solving single matching problems, as we mentioned above.

In the remainder of this paper we describe in details our graph matching approach.

2.2 Algorithm

We consider at the beginning only one graph $G^I = (V^I, E^I, D^I)$ of the size n_1 . Assume, that we know a partition of the node set V^I of this graph into m_1 non-overlapping clusters based on some rule: $V^I = \cup_{k=1}^{m_1} V_k^I$, where $V_{k_1}^I \cap V_{k_2}^I = \emptyset$ for $k_1 \neq k_2$. Based on this partition the initial graph G^I is subdivided into a set of node induced subgraphs $\{G[V^k]\}_{k=1}^{m_1}$. Note, that it holds $G[V^1] \cup \dots \cup G[V^{m_1}] \subset G^I$, because edges between different subgraphs are not presented in the left union. Further, we define a mapping U between the set of graph nodes V^I and another set

of nodes $V^{Ia} = \{a_k^I\}_{k=1}^{m_1}$, which represent each subgraph $G[V_k^I]$ with a single node a_k^I . This mapping can be expressed as an matrix $U^{Ia} \in \{0, 1\}^{n_1 \times m_1}$ with elements

$$U_{ik}^{Ia} = \begin{cases} 1, & \text{if node } v_i \in V_k^I, \\ 0, & \text{otherwise.} \end{cases} \quad (2.5)$$

The new set V^{Ia} defines a node set of a new graph built on top of the other. A pair of new nodes $a_{k_1}^I, a_{k_2}^I \in V^{Ia}$ is connected with an edge, if there is at least one edge in the initial graph G^I between the corresponding clusters $V_{k_1}^I$ and $V_{k_2}^I$. The set V^{Ia} together with the set of edges between its elements and correspondence matrix U^{Ia} build a new graph $A^I = (V^{Ia}, E^{Ia}, U^{Ia})$. We will call the graph A^I an *anchor graph* of the graph G^I and its nodes *anchor nodes* or just *anchors*. The graph G^I together with its anchor graphs A^I build a two level system: the graph G^I is located on the lower (finer) level and the graph A^I on the higher (coarser) level (see Fig. 2.1).

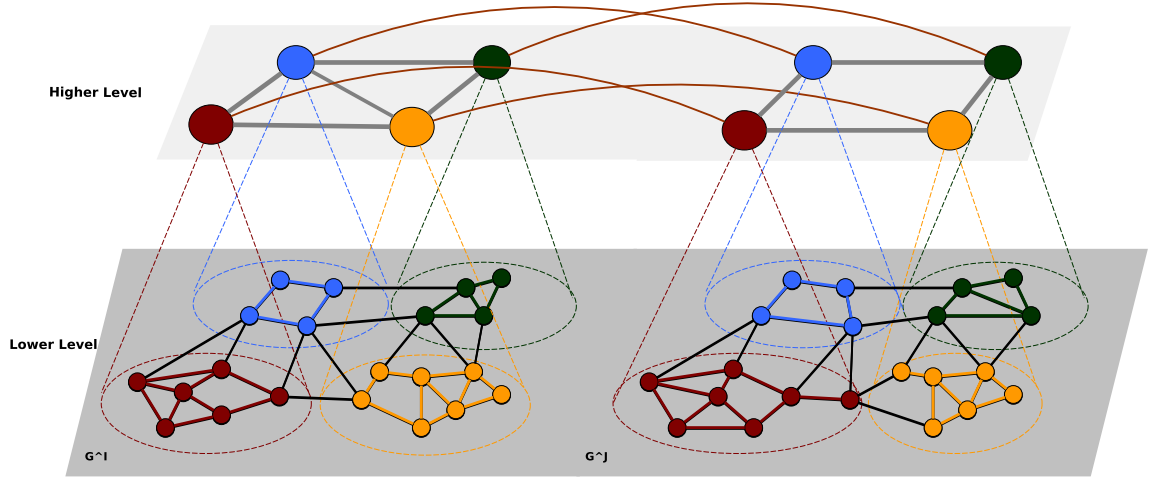


Figure 2.1: Two level framework for graph matching

We return now back to the case of two graphs $G^I = (V^I, E^I, D^I)$ and $G^J = (V^J, E^J, D^J)$, which we want to match. For each of them we build an anchor graph $A^I = (V^{Ia}, E^{Ia})$ and $A^J = (V^{Ja}, E^{Ja})$ respectively. Now instead of matching graphs G^I and G^J directly on the lower level, we may want to match first the corresponding anchor graphs. Matches between anchor nodes give us correspondences between underlying subgraphs. After that, we can perform graph matching for each pair of subgraphs completely independently. A union of local solutions from single subgraph matching problems gives us a solution of initial problem.

Why this approach can be better than direct one? As we seen from the previous

section the complexity of the considering graph matching problem depends highly on the size of initial graphs. Constructed anchor graphs are however several times smaller than initial graphs, which means, the matching algorithm on the anchor level can be performed much faster than on the lower level. The same holds for matching between the subgraphs. Obviously, the accuracy of such two level matching approach depends heavily on the partition of the initial graphs into subgraphs and on the matching quality of the anchor graphs. To make the described two level approach more robust again graph partitioning we suggest to perform described steps iteratively till convergence of the objective function (2.1). On each iteration we want to use an obtained matching between two graphs to correct graph partitioning. The algorithm is summarized below in Alg.2.

Algorithm 2: twoLevelGM(G^I, G^J, N, R, ϵ)

Input: initial graphs G^I, G^J

maximal number of iterations N

convergence parameters R and ϵ

Output: set m of correspondences between the nodes $V(G^I)$ and $V(G^J)$

```

1 construct anchor graph  $A^I$  of the graph  $G^I$ 
2 construct anchor graph  $A^J$  of the graph  $G^J$ 
3  $i=0, score_i=0$ 
4 while  $r < R$  AND  $i \leq N$  do
5    $i = i + 1$ 
6   if  $i \geq 2$  then
7      $\lfloor$  update subgraphs  $G[V_k^I], G[V_p^J], k = 1 \dots, m_1, p = 1 \dots, m_2$ 
8     match anchor graphs  $A^I, A^J$ 
9      $m_i = \emptyset$ 
10    foreach pair of matched anchors  $(a_k, a_p), a_k \in V(A^I), a_p \in V(A^J)$  do
11      match subgraphs  $G[V_k^I], G[V_p^J]$ 
12       $m_i = m_i \cup \{m_i^k\}$  /*  $m_i^k$  set of local correspondences */
13       $score_i = x^T S x$  /*  $x$  the indicator vector of the subset  $m \subseteq M$  */
14      if  $|score_i - score_{i-1}| < \epsilon$  then
15         $r = r + 1$ 
16      else
17         $r = 0$ 
18 return  $m_i$ 

```

In the following we describe in details the single steps of our approach: graph partitioning (lines ??,??), graph matching algorithm on both levels (lines ??,??), as

well as update rules of graph partitions from a previous level (line ??).

2.2.1 Anchor Graph Construction

A problem of anchor graph construction given an initial graph $G^I = (V^I, E^I, D^I)$ turns straight forward into problem of partitioning the graph G^I . During our work on this thesis we tried out different strategies for clustering nodes of a given graph. Here we present those, which were more suitable for our matching framework, however generally an arbitrary algorithm for graph partitioning can be used.

Here and further we assume that the nodes of the given fine graph G^I are located on a plane. That means, for each node we have additionally to its attribute an associated pair of coordinates and therefor can define the length of an edge as a l_2 -distance between its endpoints.

Using grid

The first algorithm we describe is the most simple one. It uses a grid with fixed number of rows r , columns c and a cell width w . The grid is placed over the graph G^I . Nodes, that are captured by a same grid cell belong to one cluster. Obviously, the number of clusters is equal to $r \times c$. We place anchor nodes in the middle of the grid cells. Two anchors are connected by an edge, if the cells they belong to have a common edge.

Algorithms based on node merge

The next considered approach creates an anchor graph A^I with a predefined number m_1 of anchors. For that we adopted a coarsening phase from multi-level graph partition algorithms [11, 57, 33, 31]. Such algorithms have generally three phases:

1. graph coarsening phase, where one creates a hierarchy of graphs by successive merging of nodes in graph on previous stage starting with initial graph;
2. graph partition stage, where the partition problem is solved exact on the coarsest level;
3. refinement phase, where solution of the coarsest level is interpolated through all levels of the hierarchy until the initial graph.

There are several types of graph coarsening algorithms. In our work we used so-called strict aggregation scheme (SAG) [11], which groups nodes of G^I in disjoint

subsets based on the strength of the edges between them. We implemented two SAG based algorithms: Heavy Edge Matching (HEM) and Light Edge Matching (LEM) [11]. Both algorithms visit nodes of the graph G^I in random order and construct an independent set of edges M of the graph. The edge selection is based on the edge weights. The HEM picks and adds into M the strongest edge adjacent to a current node v , that does not belong to the set of end nodes of edges in M . (see Alg. 3). As opposed to this, the LEM selects the weakest edge adjacent to a current node. The edges in M will be contracted, i.e. their endpoints will be replaced with a new node, that lies in the middle of a contracted edge and is connected to all neighbors of its endpoints.

Algorithm 3: HEM(G^I, m_1, N)

```

1  $i = 0, M = \emptyset$ 
2 while  $|V(G^I)| > m_1$  AND  $i \leq N$  do
3   select a random node  $v \in V(G^I) \setminus V(M)$ 
4   if  $\exists v' = \operatorname{argmax}_{u \in V(G^I) \setminus V(M)} w(v, u)$  then
5      $M = M \cup e_{vv'}$ 
6   else
7      $i = i + 1$ 
8 return  $G^I$ 

```

In our case, graph G^I is not initially weighted. To use the described coarsening methods we need to define a strength of graph edges. In case of LEM-Algorithm we set the length of an edge as its strength: $w_{vv'} = \|v - v'\|_2$. If we use HEM-Algorithm the strength of an edge is equal to $w_{vv'} = \exp(-\frac{\|v - v'\|_2}{\sigma_s^2})$ with a constant σ_s^2 .

It is clear, that one iteration of HEM or LEM reduces the number of nodes in G at most by $\lfloor \frac{n}{2} \rfloor$ nodes. To get an coarse graph with m_1 nodes the coarsening algorithm should be repeated several times.

2.2.2 Anchor graph matching

In the previous section we have described, how to construct anchor graphs $A^I = (V^{Ia}, E^{Ia}, U^{Ia})$ and $A^J = (V^{Ja}, E^{Ja}, U^{Ja})$ of given graphs $G^I = (V^I, E^I, D^I)$ and $G^J = (V^J, E^J, D^J)$ respectively. Now, we focus our attention on the problem of matching two anchor graphs with. For that, according to our problem formulation (2.1)-(2.4), we need to define a similarity matrix $S^A \in \mathbb{R}^{m_1 m_2 \times m_1 m_2}$ between

the graphs A^I and A^J , where $m_1 = |V^{Ia}|$ and $m_2 = |V^{Ja}|$. This matrix contains two types of similarities: edge similarities (non-diagonal elements) and nodes similarities (diagonal elements).

Consider two edges a pair of anchors $a_k, a'_k \in V^{Ia}$. We define the length of the edge $e_{kk'}$ between those anchors as a mean of distances between nodes in the corresponding subgraphs $G[V_k^I]$ and $G[V_{k'}^I]$. With other words:

$$L_{kk'} = \text{median}_{\substack{v_i \in G[V_k^I] \\ v_{i'} \in G[V_{k'}^I]}} \|v_i - v_{i'}\|_2, \quad (2.6)$$

where $\|v_i - v_{i'}\|_2$ is the euclidean distance between the nodes $v_i \in G[V_k^I]$ and $v_{i'} \in G[V_{k'}^I]$. Using this definition we calculate the similarity $s_E^A(e_{kk'}, e_{pp'})$ between edges $e_{kk'} \in E^{Ia}$ and $e_{pp'} \in E^{Ja}$ based on their length as it was done in the Eq. (1.10):

$$s_E^A(e_{kk'}, e_{pp'}) = \exp\left(-\frac{(L_{kk'} - L_{pp'})^2}{\sigma_s^2}\right).$$

As we already discussed in the previous chapter, the comparison of graph nodes for determining node similarity is often based on the comparison of node attributes. However, our anchor graphs A^{Ia}, A^{Ja} do not have direct attributes in contrast to the initial graphs G^I, G^J . Further we describe two ideas, which we suggest to calculate anchor similarities.

The first idea would be to assign some attributes to the anchors and proceed further in the same way, as for initial graphs. Definition of those attributes in the same way, as for nodes of original graphs, meaning taking into account only the anchor graphs and not considering underlying initial graphs, will probably not give us good results. The reason for this is, that two anchors with complete different subgraphs can get similar attributes and therefor are likely to be selected by a matching algorithm. If it happens, than the matching of the underlying subgraphs will have very low quality. This will have in turn an impact on the quality of the total matching of initial graphs. Consequently anchor attributes should incorporate the information about underlying subgraphs of original graphs. Consider an anchor $a_k \in V^{Ia}$ and its underlying subgraph $G[V_k^I] = (V_k^I, E_k^I, D_k^I)$. We suggest two classes of attributes of the anchor a_k .

- The first one uses a node attributes D_k^I of the underlying subgraph $G[V_k^I]$. For this purpose we adopted *bag of features model* [41]. We build once a common dictionary of all provided attributes in the both fine graphs by performing

k-means clustering of $D^I \cup D^J$ into C clusters. Center of the clusters represent "codewords". Each attribute of a node in V_k^I is afterwards mapped into the closest codeword. In this way, the anchor attribute $d_1(a_k) \in \mathbb{R}^C$ is defined as normalized histogram of "codewords" in corresponding subgraphs.

- The second class of descriptors should capture the geometrical structure of underlying subgraph. We define $d_2(a_k) \in \mathbb{R}^{|V_k^I| \times b}$ as a set of $|V_k^I|$ histograms $\{d_2(a_k, v)\}$ with b bins. Each histogram $d_2(a_k, v)$ represents a distribution of the length of the subgraph edges inside a small circle region around a node $v \in V_k^I$.

The similarity value between two anchors can now be determined based on the first or second type of anchor descriptors. To calculate a distance between histograms we use χ^2 statistic test [70]:

$$s_1^A(a_k, a_p) = \sum_{b_i \in B} \frac{(d_1(a_k, b_i) - d_1(a_p, b_i))^2}{(d_1(a_k, b_i) + d_1(a_p, b_i))}, \quad (2.7)$$

$$s_2^A(a_k, a_p) = \frac{1}{|V_k^I|} \frac{1}{|V_p^J|} \sum_{v \in V_k^I} \sum_{u \in V_p^J} \left(\sum_{b_i \in B} \frac{(d_2(a_k, v, b_i) - d_2(a_p, u, b_i))^2}{(d_2(a_k, v, b_i) + d_2(a_p, u, b_i))} \right), \quad (2.8)$$

where $a_k \in A^I, a_p \in A^J$ and notation $d_1(a_k, b_i)$ and $d_2(a_k, v, b_i)$ denote a value in the b_i -th bin of the corresponding histogram. Both similarities based on the anchor descriptors can be used separately or set together as a linear combination into one similarity function.

The second idea to determine similarity $s^A(a_k, a_p)$ between two anchors could be to perform graph matching of the underlying subgraphs $G[V_k^I]$, $G[V_p^J]$ and take its score as a similarity measure. This idea has a great drawback of high computational complexity, because we need to perform in summary $m_1 m_2$ local matches. However, in this case the objective function of the global matching problem and the one on the anchor level are closer related. Among other, the both should have the same trend during the optimization process. [Check](#)

2.2.3 Subgraph matching

Given two corresponding subgraphs $G_k^I = (V_k^I, E_k^I, D_k^I)$ and $G_p^J = (V_p^J, E_p^J, D_p^J)$. We use cosine similarity of the node attributes to calculate node similarity between V_k^I and V_p^J . For the pairwise edge similarity we used the same formula as in case of

anchor matching (see Eq.(1.10)), i.e.

$$s_E(e_{ii'}, e_{jj'}) = \exp\left(-\frac{(l_{ii'} - l_{jj'})^2}{\sigma_s^2}\right)$$

where $l_{ii'}$, $l_{jj'}$ are the lengths of edges $e_{ii'} \in E^I$ and $e_{jj'} \in E^J$ respectively.

2.2.4 Graph matching algorithm

Generally, we are not restricted to use one specific algorithm for subgraph and anchor graph. We selected *Reweighted Random Walks Method* (**RRWM**) [14], as it shows high matching accuracy, according to result in the original paper, and is fast. It also showed good results in finding common subgraphs of two graphs in presence of outliers. At the end of this chapter we give for completeness an overview of the method.

2.2.5 Graph partition update

Assume, we solved the graph matching problem on the higher and on the lower levels. That means, we know pairs of correspondences between the anchor nodes $m^a = \{(a_k, a_p) | a_k \in V^{Ia}, a_p \in V^{Ja}\}$ and between the nodes of the original graphs $m = \{(v_i, v_j) | v_i \in V^I, v_j \in V^J\}$. The last set is the union of the local solutions of the graph matching problem between pairs of subgraphs, as it is defined by m^a . The quality of the resulting solution m depends, as we already mentioned, in our framework not only on the quality of the graph matching algorithm, but also on the graph partitioning algorithm. The Fig. 2.2 shows an example of partition of two graphs into two classes, so that the matching result of subgraphs will be very pure for all possible combinations of anchors matches.

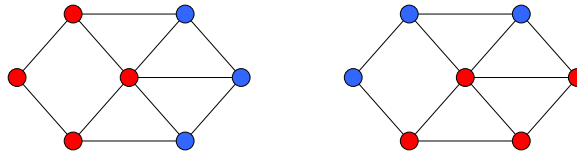


Figure 2.2: Example of bad partition of two equal graphs into two subgraphs

To cope with this problem, we formulated our method as an iterative process. After each iteration the subgraphs of initial graphs have a chance to exchange nodes with their neighbors based on the obtained solution m and improve matching quality of the next iteration. The proposed updating process consists of two major steps. In the

first step we estimate an affine transformation between matched subgraphs based on the provided local correspondences. The next step is the actual update step, where updating process uses the estimated transformations. We explain our approach on a pair of matched subgraphs $G[V_k^I] = (V_k^I, E_k^I, D_k^I)$ and $G[V_p^J] = (V_p^J, E_p^J, D_p^J)$ with obtained local correspondence set $m^{kp} = \{(v_i, v_j) | v_i \in V_k^I, v_j \in V_p^J\}$.

In the first step we want to estimate two affine transformations $T_{kp} : V_k^I \rightarrow V_p^J$ and $T_{pk} : V_p^J \rightarrow V_k^I$ from the node set of one subgraph into another and vice versa. For that we use the state-of-the-art Coherent Point Drift (CPD) algorithm by Myronenko and Song [48]. This is a probabilistic algorithm for points set registration problem, which finds a correspondences between two set of points and a transformation, that describes the mapping between sets. We choose it, as it shows a remarkable robustness against outliers and often outperforms the other popular algorithm TPS-RPM by Chui and Rangarajan [17], that we mentioned in the previous chapter. After obtaining the affine transformations T_{kp} and T_{pk} we measure a transformation error of each matched node $v_i \in V_k^I (v_j \in V_p^J)$ as a distance between its projection into other subgraph $T_{kp}(v_i) (T_{pk}(v_j))$ and its matched pair $m(v_i) = v_j \in V_p^J (m(v_j) = v_i \in V_k^I)$:

$$\begin{aligned} err(v_i) &= \|T_{kp}(v_i) - m(v_i)\|_2 \\ err(v_j) &= \|T_{pk}(v_j) - m(v_j)\|_2 \end{aligned} \tag{2.9}$$

Based on the errors of single nodes we assign an error to the estimated transformations as a measure of their quality:

$$\begin{aligned} err(T_{kp}) &= \text{median}_{v_i \in V_k^I} err(v_i) \\ err(T_{pk}) &= \text{median}_{v_j \in V_p^J} err(v_j) \end{aligned} \tag{2.10}$$

From both transformations we select the one with smallest error and replace the second with the inverse transformation of the selected one. For simplicity we preserve the notation T_{kp} and T_{pk} for the transformations related to the subgraph match (a_k, a_p) . In this way we associate with each subgraph pair with more than 3 node correspondences ¹ two affine transformations between their nodes.

In the next step, we apply estimated transformations to each subgraph of both graphs to project them into the node space of the opposite graph (see Fig. 2.3). For the subgraph $G[V_p^J]$ that means, that the transformation T_{pk} associated with

¹we need at least 3 pair of correspondences to be able to estimate an affine transformation

the anchor pair (a_k, a_p) is now casted as a mapping $T_{pk} : V_p^J \rightarrow V^I$. For the projected points $T_{pk}(v_j), v_j \in V_p^J$, we find their nearest neighbors $\bar{v}_i = NN(T_{pk}(v_j))$ in V^I . On the Fig. 2.3 the projected points $T_{pk}(v_j), v_j \in V_p^J$, are marked with bright red color. We define a new matrix $\bar{U}^{Ia} \in \mathbb{R}^{|V^I| \times m_1}$ and assign to its element $\bar{U}_{\bar{v}_i, a_k}^{Ia} = \|T_{pk}(v_j) - \bar{v}_i\|_2$ the distance between projection $T_{pk}(v_j)$ of $v_j \in V_p^J$ and its nearest neighbor $NN(T_{pk}(v_j))$ in V^I .

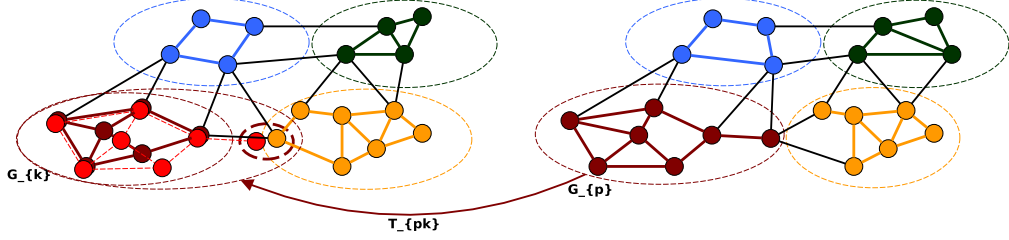


Figure 2.3: Example of an update of one (dark red) cluster

After performing the same procedure for all transformations T_{pk} we set all not-assigned elements of \bar{U}^{Ia} to infinity. If there are some lines in \bar{U}^{Ia} with all elements equal infinity, meaning that corresponding nodes in V^I were not selected as a nearest neighbor of some nodes in V^J , that we replace those lines with the corresponding lines in the current matrix U^{Ia} (see definition (2.5)).

Our update rule for the partition of the graph G^I defined by matrix U^{Ia} is:

$$U_{ik}^{Ia} = 1 \iff a_k = \underset{l=1, \dots, m_1}{\operatorname{argmin}} \bar{U}_{v_i, a_l}^{Ia} \quad (2.11)$$

With the other words, the node $v_i \in V^I$ is assigned to the anchor a_k , if \bar{U}_{v_i, a_k} is the smallest distance between v_i and projections of the nodes in the subgraphs $G[V_p^J]$ matched to the subgraph $G[V_k^I]$. Note, that unassigned nodes in V^I stay in the same cluster they were before.

The partition of the second graph is updated in the same way, as it is described above for the graph G^I . The approach is summarized in Algorithm 4.

The usage of the proposed graph partition strategy is illustrated on the Fig. 2.3. The most yellow node is likely to be included in the red partition, where in the fact is should belong to.

Algorithm 4: UpdateSubgraphs

Input: $m^a = \{(a_k, a_p) | a_k \in V^{Ia}, a_p \in V^{Ja}\}$
 $m = \{(v_i, v_j) | v_i \in V^I, v_j \in V^J\}$
Output: updated U^{Ia}, U^{Ja}

/ Step 1: assign affine transformations to each pair (a_k, a_p) */*

1 **foreach** *matched subgraph pair* (a_k, a_p) **do**

2 \lfloor calculate $T_{kp} : V_k^I \rightarrow V_p^J$ and $T_{pk} : V_p^J \rightarrow V_k^I$ using CPD algorithm

/ Step 2: calculate new matrices $\bar{U}^{Ia} \in \mathbb{R}^{|V^I| \times m_1}$, $\bar{U}^{Ja} \in \mathbb{R}^{|V^J| \times m_2}$ */*

3 **foreach** *matched subgraph pair* (a_k, a_p) **do**

4 $\lfloor \forall v_j \in V_p^J : \bar{U}_{\bar{v}_i, a_k}^{Ia} = \|T_{pk}(v_j) - \bar{v}_i\|_2$, where $\bar{v}_i = NN(T_{pk}(v_j)) \in V^I$

5 $\lfloor \forall v_i \in V_k^I : \bar{U}_{\bar{v}_j, a_p}^{Ja} = \|T_{kp}(v_i) - \bar{v}_j\|_2$, where $\bar{v}_j = NN(T_{kp}(v_i)) \in V^J$

/ Step 3: update partitions */*

6 $U_{ik}^{Ia} = 1 \iff a_k = \operatorname{argmin}_{l=1, \dots, m_1} \bar{U}_{v_i, a_l}^{Ia}$

7 $U_{jp}^{Ja} = 1 \iff a_p = \operatorname{argmin}_{q=1, \dots, m_2} \bar{U}_{v_j, a_q}^{Ja}$

8 **return** U^{Ia}, U^{Ja}

2.3 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, 35]:

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

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* [9] (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.

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* [\[9\]](#):

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

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 $D \otimes F$ *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 first term of 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 $\text{tr}(F \hat{P} D^T \hat{P}^T)$. After performing the similar calculations on the second term of the objective in (1.2) we obtain the problem (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] C. Chevalier and I. Safro. Comparison of coarsening schemes for multilevel graph partitioning. In *Learning and Intelligent Optimization: Third International Conference, LION 3. Selected Papers*, pages 191–205. Springer-Verlag, 2009.
 - [12] 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.
 - [13] 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.
 - [14] M. Cho, J. Lee, and K. M. Lee. Reweighted Random Walks for Graph Matching. *ECCV*, 2010.
 - [15] 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.
 - [16] 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.
 - [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. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 3rd edition, 2009.
 - [20] T. Cour, P. Srinivasan, and J. Shi. Balanced graph matching. In *Advances in Neural Information Processing Systems (NIPS)*, pages 313–320, 2006.
 - [21] 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.
- [22] R. Diestel. *Graph Theory, Electronic Edition 2005*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag New York, 2005.
 - [23] 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.
 - [24] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(4):619—633, 1975.
 - [25] M. Fischler and R. Elschlager. The Representation and Matching of Pictorial Structures. *IEEE Transactions on Computers*, 22(1):67–92, 1973.
 - [26] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.
 - [27] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.
 - [28] S. Gold 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.
 - [29] G. Golub and C. Van Loan. *Matrix Computations*. Last access on 28/09/2015.
 - [30] 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.
 - [31] B. Hendrickson and R. Leland. A Multi-Level Algorithm For Partitioning Graphs. In *Proceedings of the IEEE/ACM SC95 Conference*, pages 1–14, 1995.
 - [32] 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.
 - [33] G. Karypis and V. Kumar. Multilevel graph partitioning schemes. In *Proc. 24th Intern. Conf. Par. Proc., III*, pages 113–122. CRC Press, 1995.
 - [34] 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.

- [35] 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.
- [36] H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
- [37] A. Langville and C. Meyer. Deeper Inside PageRank. *Internet Mathematics*, 1(3):335–380, 2003.
- [38] 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.
- [39] M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. In *ICCV*, 2005.
- [40] M. Leordeanu, M. Hebert, R. Sukthankar, and M. Herbert. An Integer Projected Fixed Point Method for Graph Matching and MAP Inference. In *NIPS*, 2009.
- [41] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001.
- [42] Y. Lu, K. Huang, and C.-L. Liu. A fast projected fixed-point algorithm for large graph matching, 2012.
- [43] 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.
- [44] V. Lyzinski, D. L. Sussman, D. E. Fishkind, H. Pao, L. Chen, J. T. Vogelstein, Y. Park, and C. E. Pribe. Spectral Clustering for Divide-and-Conquer Graph Matching. 2011/14.
- [45] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *CoRR*, 2013.
- [46] B. Messmer and H. Bunke. A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recognition*, 32(12):1979–1998, 1999.
- [47] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemi-*

- cal Physics*, 21:1087–1092, 1953.
- [48] A. Myronenko and X. Song. Point-Set Registration: Coherent Point Drift. *CoRR*, abs/0905.2635, 2009.
 - [49] 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.
 - [50] 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.
 - [51] H. Qiu and E. R. Hancock. Graph matching and clustering using spectral partitions. *Pattern Recognition*, 39(1):22–34, 2006.
 - [52] A. Rangarajan and E. Mjolsness. A lagrangian relaxation network for graph matching. In *IEEE Trans. Neural Networks*, pages 4629–4634. IEEE Press, 1996.
 - [53] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice Hall College Div, 1977.
 - [54] K. Rose. Deterministic annealing, clustering and optimization, 1991.
 - [55] 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.
 - [56] 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.
 - [57] I. Safro, P. Sanders, and C. Schulz. Advanced coarsening schemes for graph partitioning. In *Experimental Algorithms*, volume 7276 of *Lecture Notes in Computer Science*, pages 369–380. Springer Berlin Heidelberg, 2012.
 - [58] S. Sahni. Computationally Related Problems. *SIAM J. Comput*, 3(4):262–279, 1974.
 - [59] 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.
 - [60] C. Schellewald and C. Schnörr. Probabilistic subgraph matching based on con-

- vex relaxation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3757 LNCS:171–186, 2005.
- [61] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.
 - [62] 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.
 - [63] K. Shearer, H. Bunke, and S. Venkatesh. Video sequence matching via decision tree path following. *Pattern recognition letters*, pages 479–492, 2001.
 - [64] 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.
 - [65] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *Annals of Mathematical Statistics*, 35(2):876—879, 1964.
 - [66] Y. Suh, K. Adamczewski, and K. M. Lee. Subgraph Matching using Compactness Prior for Robust Feature Correspondence. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
 - [67] 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.
 - [68] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.
 - [69] S. Umeyam. An Eigendecomposition Approach to Weighted Graph Matching Problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10, 1988.
 - [70] D. Van der Weken, M. Nachtegaal, and E. Kerre. Some new similarity measures for histograms. In *ICVGIP*, pages 441–446, 2004.
 - [71] 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.
 - [72] J. T. Wang, K. Zhang, and G.-W. Chirn. Algorithms for approximate graph

- matching. *Information Sciences*, 82(1-2):45–74, 1995.
- [73] 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.
- [74] M. Zaslavskiy. Graph matching and its application in computer vision and bioinformatics, 2010.
- [75] M. Zaslavskiy, F. Bach, and J.-P. Vert. A Path Following Algorithm for Graph Matching. In A. Elmoataz, O. Lezoray, F. Nouboud, and D. Mammass, editors, *Image and Signal Processing*, volume 5099 of *Lecture Notes in Computer Science*, pages 329–337. Springer Berlin Heidelberg, 2008.

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)