Version of 13.10.2015

Faculty of Mathematics and Computer Science

University of Heidelberg

Master thesis

in Scientific Computing

submitted by

Ekaterina Tikhoncheva

born in Moscow

2015

Application of graph matching

in

Computer Vision

This Master thesis has been carried out by Ekaterina Tikhoncheva

at the

Heidelberg Collaboratory for Image Processing

Computer Vision group

under the supervision of

Prof. Dr. Björn Ommer

Contents

Chapter 1

Graph Matching

In this chapter we introduce different forms and formulations of the graph matching problem together with some algorithms for solving them. The classification we use is based on the one introduced by Conte et al. [19]. Not all algorithms, we will present, were initially mentioned in [19], but we also do not cover all of the recent ones due to their quantity. Our focus lies especially on those, that are important for further reading of the thesis.

To begin with we refresh basic definitions and notations from 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 $E \subseteq \{\{u, v\} | u, v \in V\}$ [23]. 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.

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

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

A graph $G' = (V', E')$ is called a *subgraph* of a graph $G = (V, E)$, if $V' \subseteq V$ and $E' \subseteq E$. We use the standard notation $G' \subseteq G$ for this. A subgraph G' of G is *induced by a node subsset* $V' \subseteq V$, if $E' = \{(v_i, v_{i'}) | v_i, v_{i'} \in V'\}$. Analog, a subset $E' \subseteq E$ induces a subgraph G' of G, if $V' = \{v \in V | v \in e \text{ and } e \in E'\}$. For an induced subgraph we use the notation $G' = G[V']$ and $G' = G[E']$ [23], 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 a 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 $m : V^I \to V^J$ between the sets of nodes of two graphs. It is clear, that defined in this way the mapping m is not unique. Assume, that we have some function $S(G^I, G^J, m)$ to measure the quality of matching m. In this case, the *graph matching problem* between G^I and G^J can be defined as a problem of finding such a map $m : V^I \to V^J$, that maximizes the similarity score $S(G^I, G^J, m)$ between the graphs and has some additional constraints:

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

Based on the required properties of the mapping m the algorithms, that solve this problem, can be divided into two large groups [19]: *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 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.

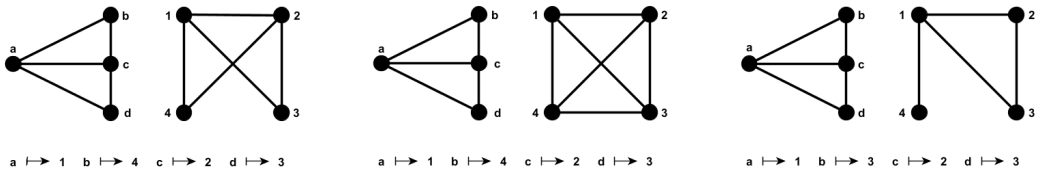
There are several forms of exact graph matching. The most known one is *graph isomorphism*: two graphs are called *isomorphic* ($G^I \simeq G^J$), if a edge preserving mapping between their nodes is bijective. This implies automatically, that two graphs should have the equal number of nodes. If it is 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*. 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.

A 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 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). In Fig. 1.1 one can see examples of different exact graph matching problems. Node correspondences are listed below each subfigure.

All problems except graph isomorphism are proofed to be $NP-$complete [30]. 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 [30, 68]. However for some special types of graphs there exist polynomial time algorithms (e.g. for trees [1, 30]).

In the following we describe several approaches for solving them.



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

Figure 1.1: Exact graph matching problems

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 the 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 [60].

One of the first algorithms, that used the described technique, is the one by Ullman [74]. 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 [42]. 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 [30].

From the other techniques we want to mention the algorithm described by

MacKay [52], which uses group theory to solve the graph isomorphism problem, and an approach based on decision trees [53, 70, 71] 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 reasons for that. On the one hand, the same object can be described by graphs with different structures. 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 problems inexactly. In this case, a strong edge preserving mapping between the nodes of two graphs is not required.

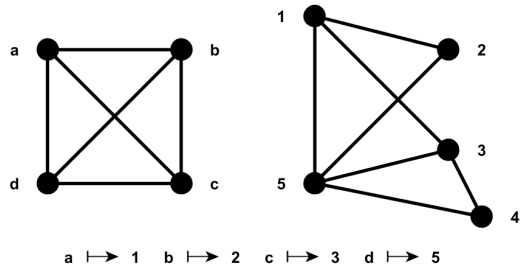


Figure 1.2: Inexact graph matching

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 an optimal inexact algorithm is not necessary faster than an 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 summarize the most common forms of the objective function of the problem (1.1).

1.4.1 Graph matching objective function

In some literature (e.g. [35, 48, 50, 63, 77, 81]), instead of defining a similarity function as it is done by Eq. (1.1), one speaks about dissimilarity between two graphs. The goal in this case is to minimize disagreement in the graph structure. Two both formulations of the problem are equivalent and can be easily transformed from one into the other.

Quadratic Optimization Problem

Let us 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, that $n_1 \leq n_2$. 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 $\{1, \ldots, n\}$. Each permutation σ can be represented by the permutation matrix $P = \{P_{ij}\}$, where

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

We denote with Π_n the set of all feasible permutations:

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

Let matrices A^I and A^J be the adjacency matrices of the graphs G^I and G^J respectively. We assume, that in case of weighted graphs, the adjacency matrices contain edge weights, instead of binary values. We can now formulate the graph

matching problem as (compare with [35, 48, 50, 63, 75, 81]):

$$P = \underset{\hat{P}\in\Pi_n}{\operatorname{argmin}} \|A^I - \hat{P}A^J\hat{P}^T\|^2 + \|D^I - \hat{P}D^J\|_2^2 \tag{1.2}$$

where $\|\cdot\|$ is the matrix Frobenius norm and $\|\cdot\|_2$ is Euclidean norm (L^2 norm). Some authors (e.g. Vogelstein et al. in [77]) use slightly different, but equivalent reformulation of it, namely:

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

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

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

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

The minimization problem (1.4) is the *quadratic assignment problem*, which is known to be $NP-$hard [9, 65].

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

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 now be rewritten as follows:

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

This formula can be also expressed in matrix form. We define an *affinity* or *similarity matrix* $S \in \mathbb{R}^{n_1 n_2 \times n_1 n_2}$, whose diagonal elements are $s_V(v_i, u_j)$ and non-diagonal elements are $s_E(e_{ii\prime}, e_{jj\prime})$. Using this matrix we obtain the following formulation of the graph matching problem as an quadratic optimization problem [15, 13, 14, 19, 31, 43, 44]:

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

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

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

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

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* [31], as they enforce one-to-one matching.

One can easily see, that the formulation (1.6)-(1.9) can be obtained from (1.4) by setting $S = (A^J)^T \otimes (A^I)^T$ with diagonal elements equal to $\text{vec}(D^I (D^J)^T)$.

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* $d_E : E^I \times E^J \to \mathbb{R}$ first and then transform it into similarity by using, for example, one of the following functions [15, 12, 13, 14]:

- $s_E(e_{ii\prime}, e_{jj\prime}) = exp(-\frac{d_E(e_{ii\prime}, e_{jj\prime})^2}{\sigma_s^2})$ (1.10)

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

where $e_{ii\prime} \in E^I$ and $e_{jj\prime} \in E^J$. The parameters σ_s and β define a sensibility of a

graph matching algorithm to the dissimilarities between the graphs.

This leads us to the question how to calculate edge dissimilarity. The most obvious way is to compare a weights of the edges, if they are provided. An other alternative could be to use the length of the edges [15, 12, 13, 14], 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.

Sometimes 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 [12, 14]:

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

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

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

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[2].

Node similarity

If a given graph has node attributes, the most natural method to measure a similarity of two nodes is to compare their attributes. In most of the seen literature, the node attributes are represented by $r-$dimensional real vectors: $D^i, D^j \subset \mathbb{R}^r$. This means, we can adopt all techniques[3] described in the 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 use metrics to measure the distance between two distributions. For example, Schellewald and Schnörr [67] used the earth mover's distance for this purpose.

[2]The cosine similarity of two vectors $x_1, x_2 \in \mathbb{R}^n$ is equal to $s_{cos} = \frac{x_1^T x_2}{\|x_1\|_2 \|x_2\|_2}$

[3]with exception of the geometric dissimilarity measure

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 one 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* [19].

Consider two attributed graphs $G^I = (V^I, E^I, D^I)$ and $G^J = (V^J, E^J, D^J)$ and matching m between subsets $\bar{V}^I, \bar{V}^J$ of V^I and V^J respectively. The edit operations on nodes can be directly defined by the mapping m [5]:

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

A edit operation of an edge is defined based on a transformation 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 $S = \{s_1, s_2, \ldots,$
$s_k\}$ denote a set of operations needed to transform the graph G^I into the graph G^J. Each edit operation $s_i, 1 \leq i \leq k$, has an assigned nonnegative cost $c(s_i)$. The cost of the whole sequence S is defined then as $\sum_{i=1}^{k} c(s_i)$. Using it and following papers [7, 78] we define a *edit distance* between the graphs G^I and G^J as follows:

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

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

The cost functions of each operation are often defined as functions of edge weights and node attributes. However their exact definition depends 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 specific conditions on the cost functions of the individual edit operations. For example, one of 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 does not depend 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 case of the error correcting graph matching. The same holds true 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 $\sum_{i=1}^{n}\sum_{i'=1}^{n}(A^I_{ii'} - A^J_{\sigma(i)\sigma(i')})^2$. A 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 so as to make the problem formulations equivalent:

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

1.5 Methods for solving inexact graph matching problems

In the following we briefly describe common approaches for solving inexact graph matching problems in the 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 tree based methods with backtracking were successfully applied to solve inexact matching problems. They were heavily 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 the exponential time 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 remaining nodes. The

algorithms in this group differ mainly in a suggested heuristics for calculation of future costs and in rules for tree traversing. The mostly used strategies for tree traversing are depth-first-search [20] and A^*-Algorithm [33].

One of the first algorithms for inexact graph matching based on three search was presented by Tsai and Fu in paper [73]. It is an optimal inexact graph matching algorithm. For further examples of tree search methods for inexact graph matching we refer to papers [7, 69, 78].

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 [54], 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 decreases the total dissimilarity of graphs, is always accepted. On the other side, a change, that increases the energy function, is accepted with some probability. This probability and number of changes per iteration is controlled by the temperature parameter T. The lower T the fewer changes are allowed and the lower the accepted probability is.

The application of simulated annealing to graph matching problems can be found in paper [35]. 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 best 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 onto 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 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 onto the discrete domain.

In 1996 Gold and Rangarajan presented a graduated assignment algorithm for graph matching (GAGM) [31]. It is an iterative algorithm, which uses *deterministic annealing*[4] to solve the graph matching problem formulated as in (1.6). The authors use Taylor series approximation to relax the initial quadratic assignment problem to a linear assignment problem. The two-way constraints are enforced at each iteration by converting a found continuous solution onto a double stochastic matrix by applying exponentiation and Sinkhorn normalization [72]. This technique is called *soft-assign*. The deterministic annealing was also used in [59] to solve Lagrangian Relaxation of the problem (1.2) and in the robust point matching algorithm with thin-plane spline (RPM-TPS) [18]. The solution obtained by this method is however not necessary optimal and the behavior of the algorithms depends highly on the selected parameters, especially those, which control the annealing schema.

Another iterative algorithm was proposed in paper [44] 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 the *Frank-Wolfe algorithm* [29] adapted to the graph matching problem. This algorithm first finds the best search direction by solving an linear assignment problem. The last arises as in GAGM by the maximization of the first-order Tailor series approximation of the objective function. The maximization is performed in the discrete domain using the Hungarian algorithm [40]. 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

[4] Deterministic 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 [61].

in [77], also uses the Frank-Wolfe algorithm to solve a continuous relaxation of the problem (1.3). However FAQ performs completely in a continuous domain and has a third step, which maps a found continuous solution back onto discrete domain. Unlike IPFP, the FAQ algorithm does not use the similarity matrix S. This gives it a big advantage as less space is required. Therefore it is possible to apply the algorithm to big size graphs[5].

Another approach, that uses the Frank-Wolfe algorithm, is the PATH-Algorithm [80]. The algorithm first finds the global optimum of the Lagrangian relaxation of the minimization problem (1.3). This is done by applying the Newton method [3]. Afterwards the found solution is projected back onto the discrete domain by solving a sequence of convex and concave problems with the Frank-Wolfe algorithm.

The fast projected fixed-point algorithm (FastPFP) proposed by Lu et al. [48] is very close to IPFP. This algorithm uses the *projected fix point method* to find a solution of a continuous relaxation of the problem (1.2). Compared to IPFP the new algorithm uses projections onto a continuous domain, instead of a 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, does not have a memory issues because it does not use a similarity matrix S. Also it 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 the graph matching problem was suggested by Schellewald and Schnörr [67]. They formulated the graph matching problem as a regularized bipartite matching problem [23], 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 onto a 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 the initial problem [21]. Consequently, the algorithm is not suitable for big graphs.

[5]Evaluation results in the paper include test with graphs with up to 1000 nodes

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, as they are invariant to the node permutation[6].

One of the first algorithms, which uses spectral techniques, is the one by Umeyama
[75]. 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 also want mention, that this algorithm is highly limited in its application due to its requirements. We also want to mention, that it works only with graph geometry and does not consider attributes of nodes and edges.

The more recent algorithm in this group of methods is called spectral matching (SM) [43]. The algorithm works with the general formulation (1.6). It relaxes both two-way and integer constraints (1.7)-(1.9) and obtain the optimal solution of this relaxation by taken the principal eigenvector of the matrix S. This is done by using *the power iteration method* [32]. 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 [24]. The authors suggest to consider similarities between tuples of nodes and not only pairs to improve the graph matching quality. They formulate a new problem using hyper-graphs and tensor notation. The solution strategy is however the same as it is in [43]: the solution is approximated with the principal eigenvector of the matrix S and

[6]Indeed, 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 $Bu = (PAP^T)u = (PA)(P^Tu) = (PA)v = P(\alpha v) = \alpha u$. So α is an eigenvalue of B.

discretized afterwards. However, the power iteration method should be adopted to the tensor based problem, because it cannot be applied directly.

Before going to the next section we want shortly to describe a max-pooling approach by Cho and Duchenne [15], which uses ideas similar to both GAGM [31] and SM [43], but replaces one of the sums in a updating step with maximum-operation. The algorithm solves the graph matching problem formulation (1.6) by relaxing its integer constraints and omitting sum constraints. To approximate a solution of $\text{argmax}_x\, x^T S x$ the max-pooling algorithm similar to GAGM uses first order Taylor expansion of the objective function. The maximization of the Taylor approximation can be done by the power method as it is done in SM. The resulting update formula of the correspondence $(v_i, v_j), v_i \in V^I, v_j \in V^J$ has the form:

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

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

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

This simple idea helps to suppress noisy entries of the outlier matches, because they have more likely low similarity values. The proposed algorithm shows very good results in presence 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 problems. 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 assigned to certain objects. Those probabilities are set based on some informations known about the objects. In terms of graph matching problems, nodes of one graph are considered as objects and nodes of another graph as labels. The 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 [28]

and Rosenfeld et al. [62]. In both cases, the 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 [17].

A first algorithm with theoretically proofed update rule is suggested by Kittler and Handcock [38]. This was later improved by Christmas at al. [17], who suggested a method to include provided node attributes and edge weights into the update rule of the labeling probabilities and not only in the initialization step.

Later algorithms formulate the graph matching problem as Maximum A Posteriori probability (MAP) estimation. For example, the matching process in paper [79] is a process of node exclusion and insertion, so that the MAP rate monotonically increases in each iteration. This technique is suggested to effectively cope with 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 constraints 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, but is slow.

Another interesting idea is presented in paper [49]. Given two graphs, nodes of the first graph are considered as an observed data and the nodes of the second graph as hidden random variables. In this formulation the graph matching problem is solved using the *Expectation-Maximization algorithm* (EM) [22]. A similar algorithm is presented in the recent paper [66]. 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 information[7] into the matching process, whereby the algorithm from [49] uses only geometrical information. Both algorithm however are not applicable to big graphs due to unlimited increasing of the formulated density function with the size of a graphs [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 the

[7]Structural relations between nodes are given through the adjacency matrix of a graph. Geometrical relations are represented as relative position of the nodes with respect to each other [66].

calculation of the density function.

Methods using clustering techniques

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

Minsu Cho [12] 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 are defined as in Eq. (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 detect them using threshold value of the cluster size.

Another algorithm based on the hierarchical clustering is suggested by Carcassoni and Hancock [10]. Its main idea is to provide additional constraints, that can be used to improve the robustness of a graph matching algorithm against graph deformations. This was successfully achieved by using spectral methods to build the clusters. The S eigenvectors associated with the first S largest eigenvalues are selected as a cluster centers. After performing the clustering, a probabilistic matching algorithm is applied to match first the clusters and then the nodes inside the clusters.

A further benefit of using clustering methods is 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 Handcock [58]. They suggest to use a 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 neighborhoods[8]. To build those they use Fiedler vector (eigenvector associated with the second smallest eigenvalue of a graph Laplacian matrix, see [25]). 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 the structure of initial graphs. The last is 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 the reduced problem size faster. A solution of the whole problem is then obtained by combining local solutions. The great benefit of such an approach is the ability to parallelize the computation process, which can lead to a significant time reduction.

The described idea is followed by Lyzinski et al. [50]. 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, what means 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, nodes 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 graphs with $20000 - 30000$ nodes.

1.6 Discussion

In this chapter we have introduced graph matching problem and talked about different approaches for solving it. We have seen, that this problem can have many variations depending on the required properties of matching. One distinguishes often two main groups: exact and inexact graph matching problems. The first group is characterized by searching for a structure preserving mapping between two graph. This requirement is however often too strict for practical applications, where some variations in the graph structure are normal. For this reason we concentrated ourself on the inexact graph matching problems, whose aim is to find

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

the best possible matching between two graphs. We have shown different ways to formulate inexact graph matching problem (e.g. as quadratic assignment, least squares problems or using graph edit distance) and how different formulations are connected to each other. At the end we have given an overview of existing algorithms for solving graph matching problems. We used the extensive survey by Conte et al. [19] as a baseline and extended it further with the recent works in the field. The most of existing algorithms were developed for matching relative small graphs (often, only up to 100 nodes) and their direct application to the bigger graphs is sometimes impossible due to polynomial increase of time and storage demand. In the next chapter we describe a framework, which allows an application of existing graph matching algorithms for matching bigger graphs.

Chapter 2

Two level graph matching

In this chapter we describe our novel approach for graph matching. Our aim however was not to develop a new matching algorithm, but to propose a framework, which would help to solve problems, where the direct application of existing matching algorithms is impossible due to memory requirements or runtime. This is often the case when a graph matching problem is formulated using a similarity matrix between graphs (see Eq. (1.6)). The main idea of our approach is based on the *divide and conquer* technique [20], which is well known from its application for 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 subproblems are created in such a way, that they can be solved without great effort. A resulting solution is then combined from local solutions of single subproblems. A disadvantage of such an approach is however, that a runtime improvement is sometimes paid with a drop in the accuracy. Due to that, one want to have a trade off between speed up and accuracy. What is more important depends on a particular problem.

Several existing algorithms share similar ideas. Those which are closest to our approach were described in the previous chapter under the group of algorithms based on clustering techniques. Below we review them shortly again to point out, that none of them is completely repeated by our framework.

This chapter is organized as follows. First of all we formulate a 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 the 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 $V^{I(J)}$, $E^{I(J)}$, $D^{I(J)}$ 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 $r-$dimensional real vectors: $D^I, D^J \in \mathbb{R}^r$.

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

$$\underset{x}{\operatorname{argmax}}\, x^T S x \tag{2.1}$$

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

$$\sum_{i=1\ldots n_1} x_{ij} \leq 1 \tag{2.3}$$

$$\sum_{j=1\ldots n_2} x_{ij} \leq 1 \tag{2.4}$$

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 set M. This 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 Eq. (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 elements $S_{kk}, k = 1, \ldots, n_1 n_2$, contain similarity measurements of the node pairs $(v_i, v_j) \in M$. The non-diagonal elements measure similarity of edges between two pairs of matched nodes. Our aim is to find 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 the required memory demands. 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 the 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 issues of the problem formulation as defined in Eqs. (2.1)-(2.4), but also reduces the algorithmic complexity of many algorithms, which highly depend on the number of possible matches (e.g. [15, 13, 14, 43]).

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

The third possibility is to replace the initial problem of graph matching by a set of smaller subproblems, that arise by partitioning given graphs into subgraphs and matching those subgraphs. It means, that the matrix S is divided into blocks, where each block represent a similarity matrix between two subgraphs. Thereby the similarities of edges, whose nodes belong to different subgraphs after problem splitting, 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 necessary reduce the time complexity of the initial problem, as selection of correspondences between subgraphs means in the worst case their pairwise matching in all possible combinations. Otherwise, further information will be lost. Despite the mentioned drawbacks, the single subgraph matching problems can be eventually parallelized, that still makes the approach attractive to be applied on big graphs.

In the framework for graph matching, that we describe in detail below, we use the third of the described techniques. We divide 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 an existing matching algorithm. A graph partitioning is performed only at the initialization step, but after each iteration subgraphs have a chance to

exchange nodes on their borders.

To our best knowledge the described method was not published before. Especially, we haven not seen an iterative graph matching algorithm based on graph clustering so far, which updates 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. [50] uses graph partitioning to parallelize a semi-supervised graph matching problem, where some correspondences between graph 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 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 an unsupervised matching.

An idea similar to our one to use graph partition for graph matching in unsupervised cases was used by Carcassoni and Hancock [10], Qui and Handcock [58] and recently by Nie et al. [56]. From them only the algorithm by Nie et al. [56] considers the same maximization problem as we do. The two other algorithms formulate the 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 direct neighborhoods of nodes. The resulting partition can be overlapping [56] or not [58]. Our algorithm and the one in the paper of Carcassoni and Hancock [10] consider the more general case, where a graph partition is given by a disjoint set of arbitrary subgraphs of a graph. Finally, similar to our approach Qui and Handcock [58] use the extracted graph partitioning to create a new graph, whose nodes represent clusters of the initial graph. They call this process graph simplification. However their approach quite differs from ours, because of the special definition of clusters, another problem formulation and a different approach for solving the single matching problems, as we mentioned in the previous chapter.

In the remainder of this chapter we describe in detail our graph matching framework.

2.2 Two level graph matching algorithm

For simplicity we consider at the beginning only one graph $G^I = (V^I, E^I, D^I)$ of 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 $\{G[V_k^I]\}_{k=1}^{m_1}$. Note, that it holds $G[V^1] \cup \cdots \cup G[V^{m_1}] \subset G^I$, 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 $V^{Ia} = \{a_k\}_{k=1}^{m_1}$, where a single node a_k represents a subgraph $G[V_k^I]$. This mapping can be expressed as an matrix $U^{Ia} \in \{0,1\}^{n_1 \times m_1}$ with elements

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

The new set V^{Ia} defines a node set of a new graph built on top of the other one. A pair of new nodes $a_{k_1}, a_{k_2} \in V^{Ia}$ is connected with an edge if and only 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} builds 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 graph A^I builds 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).

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 $A^I = (V^{Ia}, E^{Ia}, U^{Ia})$ and $A^J = (V^{Ja}, E^{Ja}, U^{Ja})$ respectively. Now instead of matching graphs G^I and G^J directly on the lower level, we 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 matched subgraphs independently. A union of local solutions from the single subgraph matching problems gives us a solution of the initial problem. More formally, we replace the objective function of the

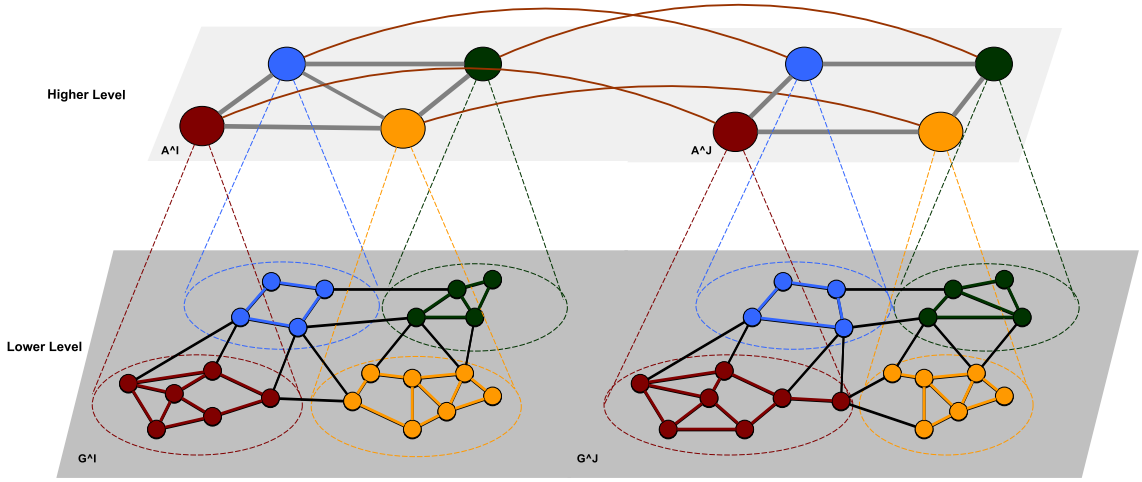


Figure 2.1: Two level framework for graph matching

initial matching problem (see Eq. (2.1), (1.5)) with the new function:

$$S(G^I, G^J, m|m^a) = \sum_{\substack{v_i, v_{i'} \in V_k^I \\ v_j, v_{j'} \in V_p^J \\ m^a(a_k)=a_p}} s_E(e_{ii'}, e_{jj'}) + \sum_{\substack{v_i \in V_k^I \\ v_j \in V_p^J \\ m^a(a_k)=a_p}} s_V(v_i, v_j), \tag{2.6}$$

where $S(G^I, G^J, m|m^a)$ denotes the similarity of the initial graphs given the matching $m^a = \{(a_k, a_p)|a_k \in A^I, a_p \in A^J\}$ of the corresponding anchor graphs or, which is the same, the correspondences $\{(V_k^I, V_p^J)|V_k^I \subset G^I, V_p^J \subset G^J\}$ between the subgraphs of the initial graphs.

Why can this approach be better than the direct one? As we cab see from the previous section the time and space complexity of the considering graph matching problem depends highly on the size of initial graphs. Constructed anchor graphs are however several time smaller than the initial graphs, which means, the matching algorithm on the anchor level can be performed much faster and without high memory demand compared to the lower level. The same holds true for matching between the subgraphs. Obviously, the accuracy of such an two level matching approach depends heavily on the partition of the initial graphs into subgraphs and on the quality of a graph matching algorithm on both the higher and lower levels. We concentrate ourself on the first of this two critical moments and use an existing algorithm for graph matching. To make the described two level approach more robust against graph partitioning we suggest to use an obtained matching between two graphs to correct their partitions. The

overall procedure (matching on the higher level, matching on the lower level and partition updating) is then repeated iteratively till convergence of the objective function (2.1). The algorithm is summarized below in Alg.1.

Algorithm 1: 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 update subgraphs $G[V_k^I], G[V_p^J], k = 1 \ldots, m_1,\ p = 1 \ldots, m_2$
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 $m_i \subseteq M$ */
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 detail the single steps of our approach: graph partitioning (lines 1,2), graph matching algorithm on both levels (lines 8,11), as well as the update rule for current graph partitions (line 7).

2.2.1 Anchor Graph Construction

The problem of anchor graph construction for initially given graph $G^I = (V^I, E^I, D^I)$ turns straight forward into a 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 therefore can define the length of an edge as a L^2-distance between its endpoints. The reason for this assumption is given by practical applications such as image or object recognition.

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, 64, 37, 34]. 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 phase, 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 E' of the graph. The edge selection is based on the edge weights. The HEM picks and adds into E' the strongest edge adjacent to a current node v, that does not belong to the set of end nodes of edges in E' (see Alg. 2). As opposed to this, the LEM selects the weakest edge adjacent to a current node. The edges in E' 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 2: HEM(G^I, m_1, N)

```
1 i = 0, E' = ∅
2 while |V(G^I)| > m_1 AND i ≤ N do
3     select a random node v ∈ V(G^I) \ V(M)
4     if ∃v' = argmax_{u∈V(G^I)\V(M)} w(v,u) then
5         E' = E' ∪ e_{vv'}
6     else
7         i = i + 1
8 return G^I
```

In our case, the 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-Algotihm 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 $w_{ii'} = exp(-\frac{\|v-v'\|_2}{\sigma_s^2})$ 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 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. For that, according to our problem formulation (see Eqs. (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 node

similarities (diagonal elements).

Let us consider 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 median of distances between nodes in the corresponding subgraphs $G[V_k^I]$ and $G[V_{k'}^I]$. With other words:

$$L_{kk'} = \underset{\substack{v_i \in G[V_k^I] \\ v_{i'} \in G[V_{k'}^I]}}{\text{median}} \|v_i - v_{i'}\|_2. \tag{2.7}$$

Using this definition we calculate the similarity $s_E^A(e_{kk'}, e_{pp'})$ between the edges $e_{kk'} \in E^{Ia}$ and $e_{pp'} \in E^{Ja}$ based on their length as it was done in Eq. (1.10):

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

As we already discussed in the previous chapter, a natural way to define node similarities is to compare their attributes. However, our anchor graphs A^{Ia} and A^{Ja} do not have direct attributes in contrast to the initial graphs G^I, G^J. Further we describe two ideas, how to work around this problem.

The first idea is to assign some attributes to the anchors and proceed further in the same way, as for the initial graphs. However, if we define those attributes based only on the anchor graphs without involving the provided information about underlying initial graphs, the overall matching result can be corrupted. The reason for this is, that an anchor in one graph can get a similar attribute, as an anchor from the other graph, although those anchors represent different subgraphs in the original graphs. If they will be selected by the matching algorithm, 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 node attributes D_k^I of the underlying subgraph $G[V_k^I]$. For this purpose we adopted the *bag of features model* [45]. We build once a common dictionary of all provided attributes in the two fine graphs by performing *k-means clustering* of $D^I \cup D^J$ into C clusters. The centers of the clusters represent "codewords". Each attribute of a node in V_k^I is

afterwards mapped onto 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 the corresponding subgraphs.

- The second class of attributes 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 attributes. To calculate a distance between histograms we use χ^2 statistic test [76]:

$$s_1^A(a_k, a_p) = \sum_{b_i \in B} \frac{(d_1(a_k, b_i) - d_1(a_p, b_i))^2}{(d_1(a_k, b_i) + d_1(a_p, b_i)}, \tag{2.8}$$

$$s_2^A(a_k, a_p) = \frac{1}{|V_k^I|} \frac{1}{|V_p^J|} \sum_{v \in V_k^I} \sum_{u \in V_p^J} \Big(\sum_{b_i \in B} \frac{(d_2(a_k, v, b_i) - d_2(a_p, u, b_i))^2}{(d_2(a_k, v, b_i) + d_2(a_p, u, b_i))} \Big), \tag{2.9}$$

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 defined similarities 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 is 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 total $m_1 m_2$ local matches. However, in this case the objective function of the matching problem on the lower level and the one on the anchor level are closer related. For example, in case when we do not consider edge similarities between anchors and work only with anchor similarities, the objective score on the higher level represent the lower bound of the objective function (2.6) on the lower level. Indeed, this is true, because the objective function (2.6) consist in this case of two parts: the objective function on the higher level and additional summands, which correspond to the similarities of edges between clusters.

2.2.3 Subgraph matching

Given are 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 we used for the anchor matching (see Eq.(1.10)), i.e.

$$s_E(e_{ii'}, e_{jj'}) = exp(-\frac{(l_{ii'} - l_{jj'})^2}{\sigma_s^2}) \tag{2.10}$$

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 matching. In this thesis we used *Reweighted Random Walks Method* (**RRWM**) [13], as it shows high matching accuracy according to results in the original paper and additionally is fast. It also shows good results in finding common subgraphs of two graphs in presence of outliers. For reasons of completeness we give an overview of this algorithm in appendix B. However, we do not provide theoretical justification of the algorithm steps and refer a reader to the original paper for that.

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 a partition of two equal graphs, so that the matching result of subgraphs will be very pure for all possible combinations of anchors matches.

To cope with such problems, we formulated our method as an iterative process. After each iteration the subgraphs of the initial graphs have a chance to exchange nodes with their neighbors based on the obtained solution m and im-

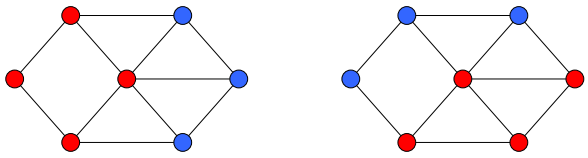


Figure 2.2: Example of bad partition of two equal graphs. Since clusters are very different, their matching in all possible combinations will always have low quality.

prove the 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 the 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 the 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 $T_{kp} : V_k^I \to V_p^J$ and $T_{pk} : V_p^J \to V_k^I$ 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 [55]. This is an probabilistic algorithm for the points set registration problem, which finds correspondences between two set of points and a transformation that describes the mapping between the sets. We choose it, because it shows a remarkable robustness against outliers and often outperforms the other popular algorithm TPS-RPM by Chui and Rangarajan [18], which we mentioned in chapter 1. After obtaining the affine transformations T_{kp} and T_{pk} we measure a transformation error of each matched node $v_i \in V_k^I (v_j \in V_k^J)$ as a distance between its projection into the other subgraph $T_{kp}(v_i)(T_{pk}(v_j))$ and its matched pair $m(v_i) = v_j \in V_p^J$ $(m(v_j) = v_i \in V_k^I)$:

$$\begin{aligned} err(v_i) &= \|T_{kp}(v_i) - m(v_i)\|_2 \\ err(v_j) &= \|T_{pk}(v_j) - m(v_j)\|_2 \end{aligned} \tag{2.11}$$

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

$$\begin{aligned} err(T_{kp}) &= \underset{v_i \in V_k^I}{\text{median}}\, err(v_i) \\ err(T_{pk}) &= \underset{v_j \in V_p^J}{\text{median}}\, err(v_j) \end{aligned} \tag{2.12}$$

From both transformations we select the one with the 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 pair of matched subgraphs, that have at least 3 provided node correspondences[1], two affine transformations between their nodes.

In the next step, we apply the 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 the anchor pair (a_k, a_p) is now casted as a mapping $T_{pk} : V_p^J \to V^I$. 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. In 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 elements $\bar{U}^{Ia}_{\bar{v}_i, a_k} = \|T_{pk}(v_j) - \bar{v}_i\|_2$ the distance between the projections $T_{pk}(v_j)$ of $v_j \in V_p^J$ and their nearest neighbor $NN(T_{pk}(v_j))$ in V^I.

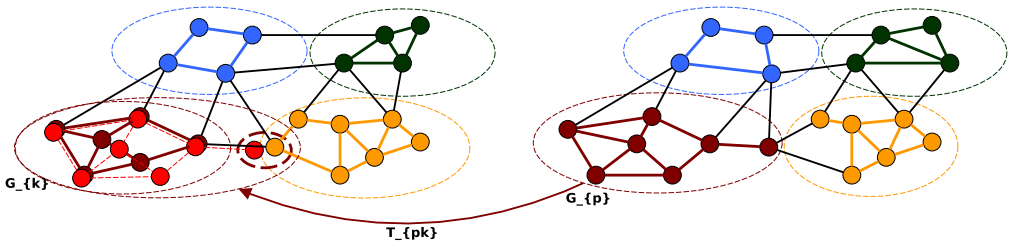


Figure 2.3: Example of the graph partition update rule

After performing the same procedure for all transformations T_{pk} we set all not-assigned elements of $\bar{U}^{Ia}$ to infinite. If there are some lines in $\bar{U}^{Ia}$ where all elements equal are infinite, meaning that the corresponding nodes in V^I are not selected as nearest neighbors of the projections of the nodes in V^J, then 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 the matrix U^{Ia} is:

$$U^{Ia}_{ik} = 1 \iff a_k = \underset{l=1,\ldots,m_1}{\operatorname{argmin}} \bar{U}^{Ia}_{v_i, a_l} \tag{2.13}$$

[1]We need at least 3 pairs of correspondences between two sets of points to be able to estimate an affine transformation between them.

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 where they were before. It also can happen, that one note on different iterations will be assigned to different clusters. This can be often the case on the border between two good estimated clusters. To prevent unnecessary jumps, we introduce a local memory of each node, where it saves information about clusters it belonged to and associated value of the matrix $\bar{U}^{Ia}$. We assign a node to a new cluster only when a new assignment is better than the previous ones.

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 3.

Algorithm 3: 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) `*/`

foreach *matched subgraph pair* (a_k, a_p) **do**
  calculate $T_{kp} : V_k^I \to V_p^J$ and $T_{pk} : V_p^J \to V_k^I$ using CPD algorithm

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

foreach *matched subgraph pair* (a_k, a_p) **do**
  $\forall v_j \in V_p^J : \quad \bar{U}^{Ia}_{\bar{v}_i, a_k} = \|T_{pk}(v_j) - \bar{v}_i\|_2$, where $\bar{v}_i = NN(T_{pk}(v_j)) \in V^I$
  $\forall v_i \in V_k^I : \quad \bar{U}^{Ja}_{\bar{v}_j, a_p} = \|T_{kp}(v_i) - \bar{v}_j\|_2$, where $\bar{v}_j = NN(T_{kp}(v_i)) \in V^J$

`/* Step 3: update partitions */`

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

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

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

The usage of the proposed graph partition strategy is illustrated in Fig. 2.3. After performing the update procedure, the most left yellow node is likely to be included in the red partition, where in fact it should belong to.

2.2.6 Complexity

We would like to investigate the asymptotic computational complexity of our two level graph matching approach. For simplicity, we assume, that the complexity of a selected graph matching algorithm is $\mathcal{O}(f_{GM}(n_1, n_2))$, where n_1 and n_2 are the size of the graphs we want to match.

The preprocessing step of our approach consists of two stages: initial graph partitioning and creation of the codebook of the node attributes. Note that the presence of the second stage depends on the selected strategy of the anchor graph matching. The complexity of the graph partitioning is in the worst case equal to $\mathcal{O}(n_2^2)$[2]. Indeed, for one graph with n nodes and m anchors it is equal to $\mathcal{O}(nm)$ (using grid approach) or $\mathcal{O}((n-m)\Delta(G))^3$ (HEM or LEM algorithm). The complexity of the codebook creation is determined by the complexity of the clustering algorithm, which in case of Lloyd's k-means algorithm [46] comes to $\mathcal{O}((n_1+n_2)rC)$ per iteration, where r is the dimension of the node attribute vectors and C the number of clusters. After joining those two results we obtain that the complexity of the preprocessing step is

$$\mathcal{O}(n_2^2 + (n_1+n_2)rCi), \tag{2.14}$$

where i is the maximum number of iterations of the k-means clustering.

The main loop of our two level graph matching approach consists of three steps. The first one is the anchor graph matching, whose complexity is $\mathcal{O}(f_{GM}(m_1, m_2))$ plus the complexity for initialization of the similarity matrix. In case, when we assign attributes to the anchors to calculate node similarity between two graphs, the complexity of the initialization can be approximated by $\mathcal{O}(m_1^2 m_2^2)$. The complexity of the subgraph matching step afterwards results in $\sum_{(a_k,a_p)\in m^a} \mathcal{O}(f_{GM}(|V_k^I|, |V_p^J|))$. The complexity of the two first steps in this case is equal to

$$\mathcal{O}(m_1^2 m_2^2) + \mathcal{O}(f_{GM}(m_1, m_2)) + \sum_{(a_k,a_p)\in m^a} \mathcal{O}(f_{GM}(|V_k^I|, |V_p^J|)). \tag{2.15}$$

In the case, when we use graph matching to determine the similarity between

[2] We assumed $n_1 \leq n_2$

[3] We denote with $\Delta(G)$ the maximal degree of nodes in V, where degree of a node is equal to the number of its incident edges [23].

anchors, the complexity of both anchor and subgraph matching is equal to

$$\sum_{(a_k,a_p)\in M^a} \mathcal{O}(f_{GM}(|V_k^I|,|V_p^J|)), \quad M^a = \{(a_k,a_p)|a_k \in V^{Ia}, a_p \in V^{Ja}\}, \tag{2.16}$$

because subgraph matching is already included in the anchor matching step.

At the end we only need to determine the complexity of our update rule. The CPD algorithm has linear complexity, which means we can estimate transformations between all matched subgraphs with demand $\sum_{(a_k,a_p)\in M^a} \mathcal{O}(\max(|V_k^I|,|V_p^J|))$. The actual update step afterwards has the complexity $\mathcal{O}(n_1m_1 + n_2m_2)$.

If we assume, that we can completely parallelize the computation of the sum in Eq. (2.15), that the total complexity of one iteration of our two level graph matching approach is equal to

$$\begin{aligned}\mathcal{O}(m_1^2m_2^2) + \mathcal{O}(f_{GM}(m_1,m_2)) + \mathcal{O}(f_{GM}(\max_k |V_k^I|, \max_p |V_p^J|)) \\ + \mathcal{O}(f_{GM}(\max_k |V_k^I|, \max_p |V_p^J|)) + \mathcal{O}(n_1m_1 + n_2m_2)\end{aligned} \tag{2.17}$$

or

$$\begin{aligned}\mathcal{O}(m_1^2m_2^2) + \mathcal{O}(f_{GM}(m_1,m_2)) + \mathbf{m_2}\mathcal{O}(f_{GM}(\max_k |V_k^I|, \max_p |V_p^J|)) \\ + \mathcal{O}(f_{GM}(\max_k |V_k^I|, \max_p |V_p^J|)) + \mathcal{O}(n_1m_1 + n_2m_2)\end{aligned} \tag{2.18}$$

depending on the selected strategy for computing the similarity between anchor graphs. Let us notice that the constant m_2 in Eq. (2.18) increases the complexity of the whole algorithm, when we use subgraph matching to calculate the similarity between anchors.

However, because of the complexity of the graph matching is the most expensive (in case of RRWM $\mathcal{O}(f_{GM}(n_1,n_2)) = \mathcal{O}(n_1^2n_2^2)$), it will fully beat the complexity of other steps and we can approximate both Eqs. (2.17) and (2.18) with

$$\mathcal{O}(f_{GM}(m_1,m_2)) + \mathcal{O}(f_{GM}(\max_k |V_k^I|, \max_p |V_p^J|)) \tag{2.19}$$

and

$$\mathcal{O}(f_{GM}(m_1,m_2)) + \mathbf{m_2}\mathcal{O}(f_{GM}(\max_k |V_k^I|, \max_p |V_p^J|)) \tag{2.20}$$

respectively.

It is obvious, that the resulting complexity of our graph matching framework depends highly on the number of iterations and on the size of the anchor graphs

and subgraphs.

2.3 Discussion

We have proposed a novel framework for solving a graph matching problem formulated as an quadratic assignment problem. Our framework is based on a known matching algorithm, but helps to cope with some limitations of its application, such as big space demand and high computation time. Its core idea of the introduces framework is to decompose an initial graph matching problem into a set of subproblems. For that we partition graphs into subgraphs, perform matching to determine pairs of subgraphs and afterwards perform matching again for each pair of subgraphs. An obtained solution at the end is used to update the graph partition. The whole procedure is repeated iteratively until it converges. The complexity of the proposed method depends highly on the complexity of the used graph matching algorithm, which in its turn depends on the size of the anchor graph and the graph subgraphs. In the next chapter we evaluate the proposed framework on synthetic and real data examples.

Chapter 3

Evaluation results

In this chapter we explore the quality of the proposed two level graph matching framework (we call it further 2LevelGM) on some synthetic and real examples. The quality is measured by the matching score and accuracy of an obtained solution together with the running time needed to find it. Note, that under accuracy we understand actually recall of the graph matching algorithms (i.e. number of correct detected matches divided by the number of all correct matches), as it is done in [15, 13, 14, 24, 31, 43, 44]. To rate the usefulness of 2LevelGM we provide comparison study across different graph matching algorithms. All tests in this chapter were run on a computer with hardware unless otherwise is specified. We implemented our framework in the software package MATLAB and used 4 workers[1] to run 2LevelGM. The sources of additionally used libraries and algorithms for comparison will be referred directly in text. To make the comparison between the algorithms fair we use in all cases the same greedy assignment algorithm [43] for the discretization of an obtained continuous solution. We also include initialization and discretization steps into runtime measurement.

3.1 Synthetic data

For the first set of tests we adopted a commonly used approach of evaluation graph matching algorithms on two synthetic generated sets of points (see [15], [13], [44]). For this purpose one generates first a set $V^I \subset \mathbb{R}^2$ of n_1 standard normally distributed points on a plane: $V^I = \{v_i = (x_i, y_i) | x_i \sim \mathcal{N}(0,1), y_i \sim \mathcal{N}(0,1), i = 1, \ldots, n_1\}$. The second set V^J represents a distorted copy of the first set with $\bar{n}$

[1] For more details see http://de.mathworks.com/help/distcomp/parallel-pools.html

additional normally distributed points. The distortion is achieved by adding a normally distributed noise with 0 mean and variance σ^2 to the coordinates of the points in V^I. That means, that the set V^J consists of $n_2 = n_1 + \bar{n}$ nodes, where n_1 points have a unique pair in the V^I and are called inliers, whereby the other $\bar{n}$ points are outliers. The task is to find correspondences between points in the two sets. Obviously, this problem can be considered as a graph matching problem. For that we consider two fully connected graphs G^I and G^J with the nodes defined by the points in V^I and V^J respectively. We assume, that the graphs do not have attributes and each node in the first graph can be theoretically matched to each node in the second graph.

For the evaluation of our graph matching framework we follow the setup of the synthetic point set tests in [15, 48] and formulate four different kinds of tests. In the *first test* we set number of outliers $\bar{n}$ to zero and vary only the deformation noise σ^2. In the *second test*, we do not have deformation noise ($\sigma^2 = 0$) and compare the behavior of the different graph matching algorithms in case of increasing number of outliers $\bar{n}$. In the *third test*, we perform the outlier test in presence of deformation in the second graph. For this we fix $\sigma^2 = 0.03$ and increase iteratively the number of outliers $\bar{n}$. Finally, at the *fourth test* we consider again two graphs with the same size, but omit randomly θ percentage of edges in both graphs with increasing θ. For performing the tests we adopted the supplemental MATLAB code provided by Minsu Cho et al. to their paper [15, 16].

The first group of tests we performed works with the relatively small graphs with $100 - 150$ nodes. This was done with intend to compare the 2LevelGM with the following well known methods: MPM [15], RRWM [13], SM [43], IPFP [44][2]. The selection of the graph size was determined by the application examples of those algorithms provided in the mentioned papers. To our best knowledge, those algorithms was not applied directly to graphs with more than 150 nodes each without some additional problem simplifications (for example, reduction of the set of possible correspondences, which is difficult to achieve for non-attributed graphs). This restriction is determined mainly by the size of the dense affinity matrix S, which is used by all algorithms. The matrix S is calculated in all cases using Eq. (2.10) with $\sigma_s^2 = 0.15$, whereby value of the σ_s^2 is chosen according to the propose made in the paper [13].

[2]We used the provided implementation of those algorithms from [16].

As our initial graphs G^I and G^J are not attributed, we first consider the formulation of 2LevelGM with non-attributed anchors graphs. The similarities between anchors are calculated in this case based on the matching score of underlying subgraphs (see chapter 2.2.2). To initialize initial subgraphs we use the grid method (see chapter 2.2.1). The average matching results over 10 runs in four defined tests are illustrated in Figs. 3.1-3.4. The vertical bars at each point show the standard deviation of the measured value (accuracy, matching score and running time) for the corresponding problem settings.

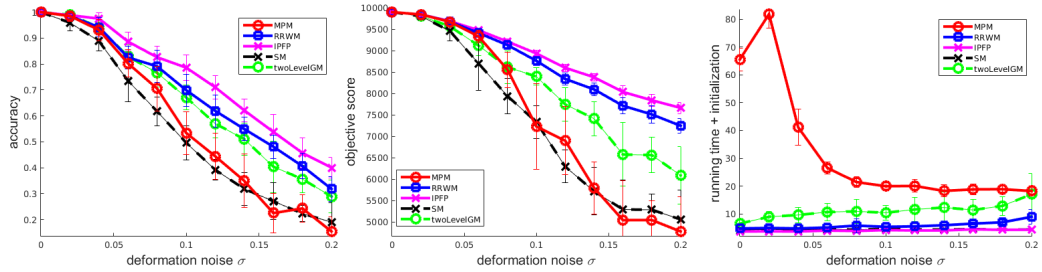


Figure 3.1: Performance of the 2LevelGM with non-attributed anchor graphs on the synthetic data: test 1 ($n_1 = 100$, $\bar{n} = 0$, $\theta = 100\%$)

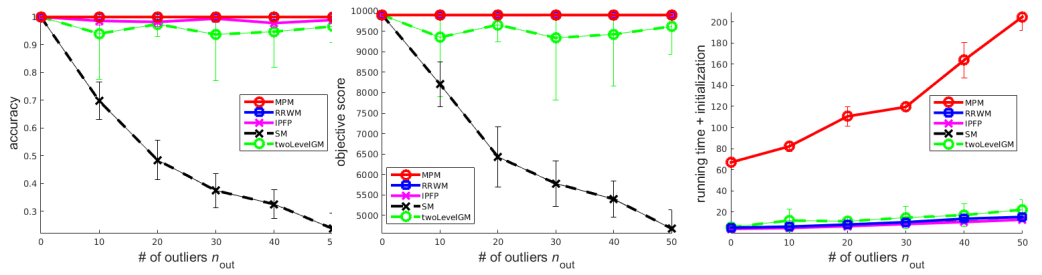


Figure 3.2: Performance of the 2LevelGM with non-attributed anchor graphs on the synthetic data: test 2 ($n_1 = 100$, $\bar{n} \in [0, 50]$, $\sigma^2 = 0$, $\theta = 100\%$)

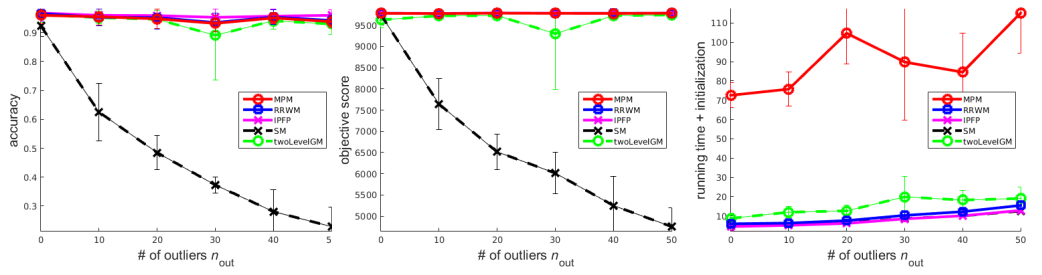


Figure 3.3: Performance of the 2LevelGM with non-attributed anchor graphs on the synthetic data: test 3 ($n_1 = 100$, $\bar{n} \in [0, 50]$, $\sigma^2 = 0.03$, $\theta = 100\%$)

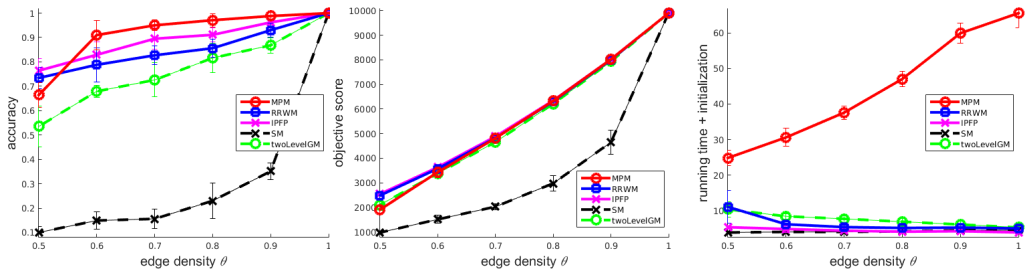


Figure 3.4: Performance of the 2LevelGM with non-attributed anchor graphs on the synthetic data: test 4 ($n_1 = n_2 = 100$, $\sigma^2 = 0$)

From this results one can see that the SM algorithm has the worst performance in all cases. The performance of 2LevelGM is a little bit unstable, what is indicated by relatively high standard deviation comparing with other algorithms. For this reason its average performance (score and accuracy) is a little bit lower that those of IPFP and RRWM, although in individual runs it is not worse. Also in running time 2LevelGM is slower than IPFP and RRWM. However, it is expected, since without anchor attributes 2LevelGM performs graph matching using RRWM on each pair of subgraphs. Although those subgraphs are smaller than the initial graphs, the overall complexity of this procedure for the relative small graphs G^I, G^J is higher than one direct application of RRWM. The best matching results are achieved in three of four tests by the MPM, which is also the slowest algorithm. However, it is outperformed by 2LevelGM in the first test (Fig. 3.4) with significantly smaller time demand[3].

Below we present the results of the same tests with the same setup for the case, when 2LevelGM works with attributed anchor graphs. For that we use the attributes, that capture geometrical structure of the subgraphs (see chapter 2.2.2). We set the size of used histograms to 35 bins and the radius R of the circle region around each node to 2. The results of comparison are presented in Figs.3.5-3.8. One can directly see the improved time performance of the 2LevelGM algorithm in all tests. The proposed algorithm also shows more stable performance in the second test, which practically solves subgraph isomorphism problem. On the other side, the algorithm seems to be more susceptible to the graph deformations (see Figs.3.5, 3.8). The reason for that lies in the selected anchor attributes, that

[3]The used implementations of 2LevelGM, SM and IPFP are purely MATLAB implementations, whereby some steps of MPM and RRWM were written using $C++$. This makes the comparison of running time between algorithms difficult.

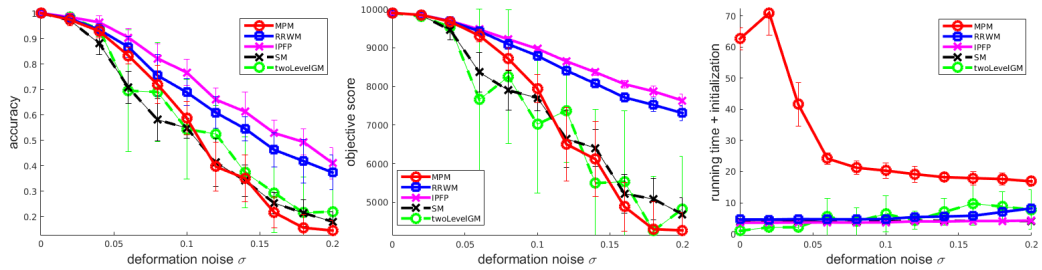


Figure 3.5: Performance of the 2LevelGM with attributed anchor graphs on the synthetic data: test 1 ($n_1 = 100$, $\bar{n} = 0$, $\sigma^2 = 0$)

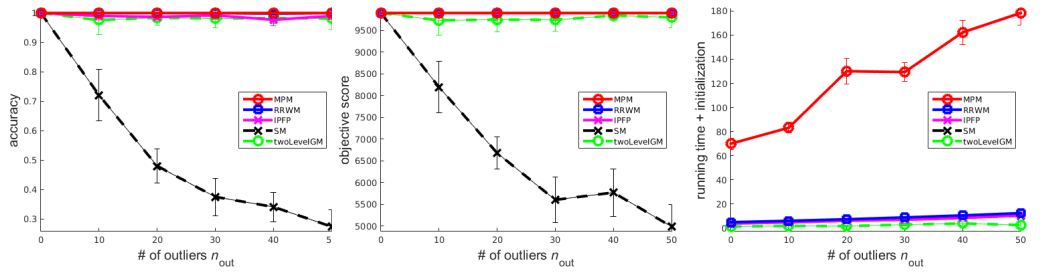


Figure 3.6: Performance of the 2LevelGM with attributed anchor graphs on the synthetic data: test 2 ($n_1 = 100$, $\bar{n} \in [0, 50]$, $\sigma^2 = 0$, $\theta = 100\%$)

are not sufficiently robust against deformations in the length of edges.

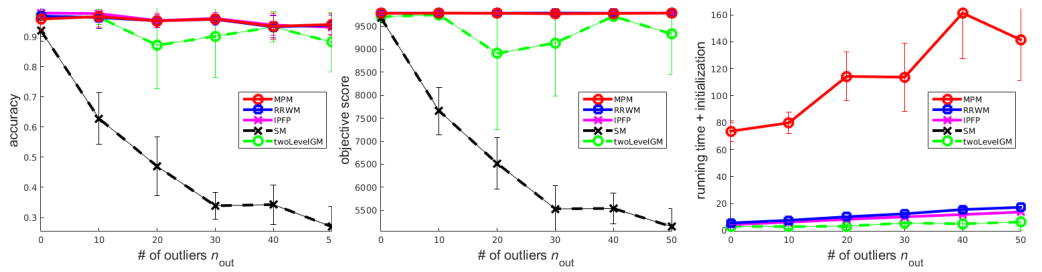


Figure 3.7: Performance of the 2LevelGM with attributed anchor graphs on the synthetic data: test 3 ($n_1 = 100$, $\bar{n} \in [0, 50]$, $\sigma^2 = 0.03$, $\theta = 100\%$)

To test the performance of our framework on bigger graphs we perform another sequence of tests and compare 2LevelGM with PATH [81, 82] and GLAG [26, 27][4] algorithms. In contrast to the graph matching problem formulation considered by 2LevelGM and previous algorithms the PATH and GLAG algorithms solve the minimization problem from Eq. (1.2). Consequently, they do not work with the affinity matrix S and can be therefore directly applied on the bigger graphs

[4]We replaced the default maximal number of iterations (30000) in GLAG with 1000.

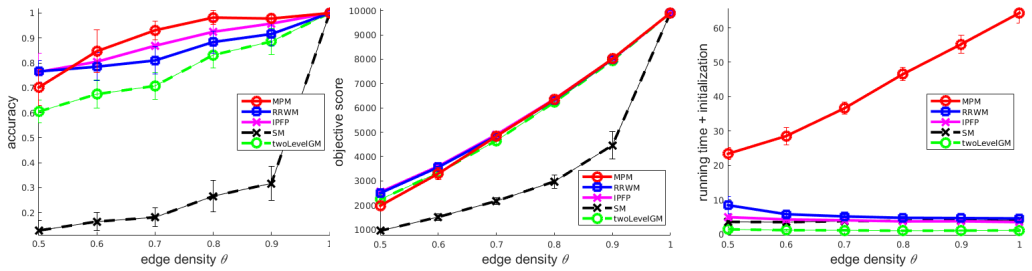


Figure 3.8: Performance of the 2LevelGM with attributed anchor graphs on the synthetic data: test 4 ($n_1 = n_2 = 100$, $\sigma^2 = 0.00$)

without necessity to reduce the set of possible candidate matches. The result of the comparison in one run can be seen in Figs. 3.9-3.11. We do not provide averaged results for this set of tests due to their high time demand.

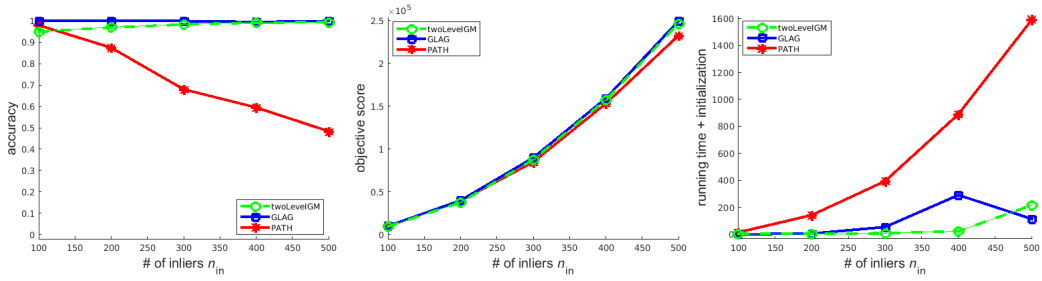


Figure 3.9: Performance comparison of 2LevelGM, FAQ, PATH on bigger graphs ($\bar{n} = 0$, $\sigma^2 = 0$, $\theta = 100\%$)

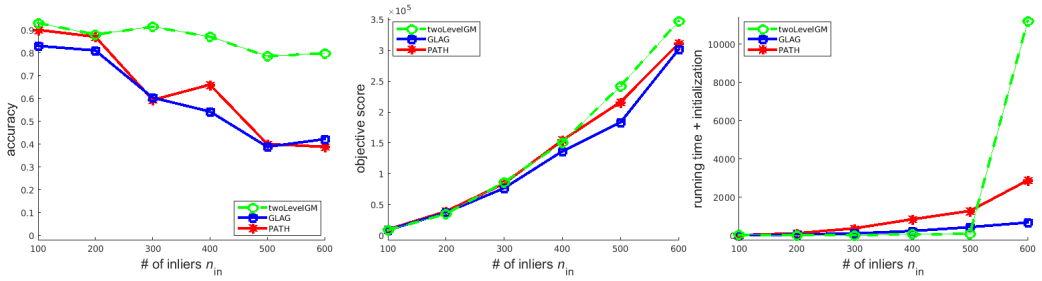


Figure 3.10: Performance comparison of 2LevelGM, FAQ, PATH on bigger graphs ($\bar{n} = 0$, $\sigma^2 = 0.03$, $\theta = 100\%$)

All performed test can be considered as instances of the graph isomorphism problem in exact (Fig. 3.9) and inexact (Figs. 3.10, 3.11) forms. We investigate the matching score, accuracy and running time of graph matching algorithms as functions of number of nodes in the initial graphs. In all cases the proposed two level graph matching framework outperforms both GLAG and PATH in matching

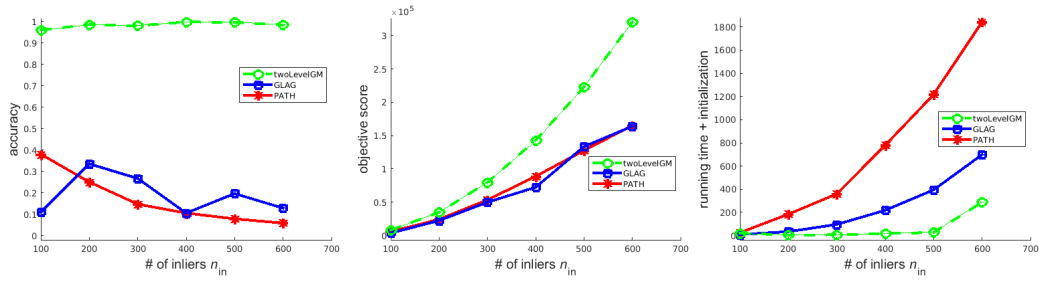


Figure 3.11: Performance comparison of 2LevelGM, FAQ, PATH on bigger graphs ($\bar{n} = 0$, $\sigma^2 = 0$, $\theta = 90\%$)

score and especially in running time. We consider the running time of the last iteration in Fig. 3.10 as an individual case and suppose, that an imbalanced graph partitioning of initial graphs causes the jump in execution time. 2LevelGM also shows the high matching accuracy in all three tests, although, we believe it is possible to improve it by solving instability issues, we saw in previous cases.

3.2 Real data test

In following sections we use the developed two level graph matching algorithms for finding correspondences between feature on a pair of images. To formulate this problem as graph matching problem we use the standard approach settled in computer vision literature [13, 14, 48, 49, 58]. For that we extract SIFT features [47] of two images at the locations found using the some feature detector (we used MSER [51]). Given the extracted features of two images we construct two attributed graphs $G^I = (V^I, E^I, D^I)$ and $G^J = (V^J, E^J, D^J)$ for each image respectively. The node sets of those graphs are defined by the provided features with there descriptors as node attributes. To connect nodes via edges one can use Delaunay triangulation [49, 58], nearest neighbors relations between nodes [66] or consider complete graphs [14, 15].

In this section we compare matching performance of 2LevelGM against simple feature matching [47] and progressive graph matching algorithm (ProgGM) [14].

3.2.1 Image Affine Transformation

The first image set we consider here can be seen as a synthetic data set of real images. It consists of images pairs, where one image in each pair is always the same and the second image represent a rotated/shifted copy of the first one (see

Fig.ref). On this simple examples we want to demonstrate the work of the update rule inside 2LevelGM (see chapter 2.2.5).

3.2.2 Real Images: House dataset

In this section,we performed feature pointmatching on theCMUHouse sequence which has been widely used in previous works [7,5] and compared with other methods. In order to assess the matching accuracy, 30 landmark feature points were manually tracked and labeled across all frames. This allows us to compare the performance of the different algorithms over a varying temporal baseline: the larger the temporal baseline between the frames, the larger the relative deformation, and the more difficult the matching.We matched all possible image pairs, total 560 pairs, spaced by 10, 20 , 30 , 40, 50 , 60 , 70, 80, 90, and 100 frames and computed the average matching accuracy per sequence gap.Graph

3.2.3 some example on Caltech-101 and MSRC

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 the cost minimization of the assignment. Let $D = (d_{kl}), F = (f_{kl}), B = (b_{ik}) \in \mathbb{R}^{n\times n}$ be real matrices that define a distances and flow between the locations, as well as the opening costs. The problem defined above can then be formulated as an integer quadratic program [9, 39]:

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

where Σ_n is a set of all possible permutations of the set $\{1,\ldots,n\}$, and is called *Koopmans-Beckmann version of the quadratic assignment problem* [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

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

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

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

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

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

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

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

From the definition of the matrix trace follows:

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

We also can write

$$A \cdot B = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}B_{ij} = \operatorname{vec}(A)^T \operatorname{vec}(B), \tag{A.5}$$

where $\operatorname{vec}(A)$ denotes the column-wise vectorization of a matrix A. Additionally, it holds [36]:

$$\operatorname{vec}(APB) = (B^T \otimes A) \operatorname{vec}(P). \tag{A.6}$$

We can now use Eq. (A.4) in Eq. (A.2) to obtain the *trace formulation of QAP* [9]:

$$P = \underset{\hat{P} \in \Pi_n}{\operatorname{argmin}} \operatorname{tr}(F\hat{P}D^T\hat{P}^T + B\hat{P}^T). \tag{A.7}$$

The objective function of the trace formulation of QAP can be further rewritten based on the properties in Eqs. (A.5) and (A.6) as follows:

$$\begin{aligned} \operatorname{tr}(F\hat{P}D^T\hat{P}^T + B\hat{P}^T) &= \operatorname{tr}(\hat{P}(F\hat{P}D^T)^T) + \operatorname{vec}(B)^T \operatorname{vec}(\hat{P}) \\ &= \operatorname{vec}(\hat{P})^T \operatorname{vec}(F\hat{P}D^T) + \operatorname{vec}(B)^T \operatorname{vec}(\hat{P}) \\ &= \operatorname{vec}(\hat{P})^T (D \otimes F) \operatorname{vec} \hat{P} + \operatorname{vec}(B)^T \operatorname{vec}(\hat{P}). \end{aligned} \tag{A.8}$$

Based on this formulation the matrix B is called sometimes *linear cost matrix* and the matrix $D \otimes F$ *quadratic costs matrix* [9, 63]. If we replace the matrix F with $-(A^I)^T$, the matrix D with $(A^J)^T$ and the matrix B with $D^I(D^J)^T$, we obtain the formulation in Eq. (1.4) from chapter 1. Analog, if we define a matrix $S = (-D \otimes F)$ with the vector $\operatorname{vec}(B)$ on the diagonal, we come to the formulation in Eq. (1.6).

It remains to show, that the problem in Eq. (1.2) is equivalent to the trace formulation of QAP (A.7). Indeed, consider the first term of the objective function in

Eq. (1.2):

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

The first and the last terms are obviously constant and thus can be ignored during optimization. Consequently, it holds:

$$\underset{\hat{P}\in\Pi_n}{\text{argmin}}\,\|A^I\hat{P} - \hat{P}A^J\|^2 = \underset{\hat{P}\in\Pi_n}{\text{argmin}} - \text{tr}((A^I)^T\hat{P}A^J\hat{P}^T). \tag{A.9}$$

We perform the same transformations with the second term:

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

Also here the first and the last terms can be ignored during optimization, so that we can write:

$$\underset{\hat{P}\in\Pi_n}{\text{argmin}}\,\|D^I - \hat{P}D^J\|^2 = \underset{\hat{P}\in\Pi_n}{\text{argmin}} - \text{tr}(D^I(D^J)^T\hat{P}). \tag{A.10}$$

The combination of the results in Eqs. (A.9) and (A.10) with the substitution $F = -(A^I)^T$, $D = (A^J)^T$ and $B = -D^I(D^J)^T$ proof the equivalence of the problems (1.2) and (A.7). That is exactly, what we wanted to show.

Appendix B

Reweighted random walks for graph matching (RRWM)

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

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

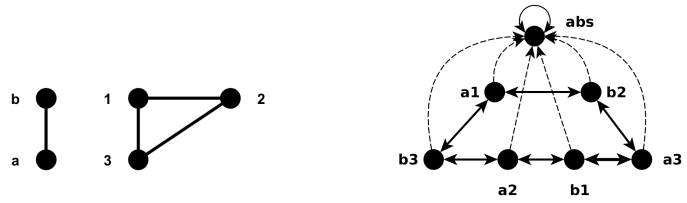
$$\text{s.t. } x \in \{0,1\}^{n_1 n_2} \tag{1.7}$$

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

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

between two graphs $G^I = (V^I, E^I, D^I)$, $G^J = (V^J, E^J, D^J)$ in a random walk view. For that purpose the authors defined 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 represents 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^{rm} as W. An example of the construction is illustrated in Fig. B.1b. Note, that we omitted contradicting edges, whose weights are equal to zero, for example $\{a1, b1\}$.

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 the matching



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

Figure B.1: Association Graph of two given graphs for Reweighted Random Walk Method (compare with [13])

criteria of the initial problem. For finding such a subset the authors adopt the page ranking algorithm based on a random walk, which is assumed to be a Markov chain [57]. To describe a Markov chain one defines a transition matrix P. An usual approach 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 the PageRank algorithm [57]. It is however not suitable for the graph matching purposes, as it treats false correspondences equal to all other correspondences. To avoid this problem Cho et al. introduced an additional absorbing node v_{abs} (see Fig. B.1b) in the Graph G^{rw}. This node represents a state, that can be reached from each node $v_k = v_{ij} \in V^{rw}$ with the probability $1 - D_{kk}/D_{\max}, D_{\max} = max_k D_{kk}$, but can not be left any more. Using $D_{\max}$ the matrix W is converted into a stochastic matrix by its multiplication with the factor $1/D_{\max}$. Summarizing, the transition matrix $P \in \mathbb{R}^{(n_1 n_2+1)\times(n_1 n_2+1)}$ is defined as

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

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

$$\left(x^{(n+1)T},\ x_{\text{abs}}^{(n+1)}\right) = \alpha \left(x^{(n)T},\ x_{\text{abs}}^{(n)}\right) P + (1-\alpha) r^T, \tag{B.2}$$

where $\mathbf{1}$ in (B.1) 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 a *reweighted jump vector*.

A Markov chain, defined by Eq. B.2 without the second term, is denoted as an

affinity-preserving random walk. The distribution $\bar{\mathbf{x}}$ of unabsorbed random walks at time n is defined as follows:

$$\bar{\mathbf{x}}_{ij}^{(n)} = P(X^{(n)} = v_{ij} | X^{(n)} \neq v_{\text{abs}}) = \frac{x_{ij}^{(n)}}{1 - x_{\text{abs}}^{(n)}}, \tag{B.3}$$

where $X^{(n)}$ is the current location of a random walker at time n. The authors call $\bar{\mathbf{x}}$ a *quasi-stationary distribution* of the absorbed Markov chain. They proof, that the distribution $\bar{\mathbf{x}}$ is proportional to the left principal eigenvector of W and can be efficiently computed with the power iteration method [32].

The second summand in Eq. (B.2) 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 an personalization approach for web page ranking [41] 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 constraints are incorporated only in the last discretization step of the algorithm, which leads to a weak local maximum.

A procedure of generation the jump vector r from a current quasi-stationary distribution $\bar{\mathbf{x}}$ consists of two steps. In the first step (*inflation*) unreliable correspondences (i.e. small values in the vector $\bar{\mathbf{x}}$) are damped and the good correspondences are boosted at the same time. The second step (*bistochastic normalization*) forces the matching constraints by transforming the matrix form of $\bar{\mathbf{x}}$ into a double stochastic matrix using the normalization scheme of Sinkhorn [72].

The described steps are summarized in Algorithm 4 below. The discretization step in line 16 can be done by using any method, which solves the linear assignment problem, i.e. the Hungarian algorithm [40] or greedy heuristic as in [43].

There are some similarities between RRWM and some algorithms, described in chapter 1. For example, the authors notice, that line 6 of Algorithm 4 can be considered as the power iteration version of SM [43]. Also the Sinkhorn normalization (line 8-11) was used in soft-assign step in [31].

The complexity of the algorithm is $\mathcal{O}(|E^I||E^J|)$ per iteration, whereby the quasi-stationary distribution $\bar{\mathbf{x}}$ in line 6 was computed with the power iteration method [32].

Algorithm 4: Reweighted Random Walks Method, compare to [13]

Input: weight matrix W, the reweight factor α, the inflation factor β
Output: distribution $\mathbf{x}$

set $W_{ij,i'j'}$=0 for all conflicting match pairs, i.e. $(v_{ij}, v_{ij'})$ and $(v_{ij}, v_{i'j})$
$D_{\max} = \max_{ij} \sum_{i'j'} W_{ij,i'j'}$
$P = W / D_{\max}$, initialize starting probability $\mathbf{x}$ as uniform
repeat
    /* Affinity preserving random walking by edges */
    $\bar{\mathbf{x}} = \mathbf{x}^T P$
    /* Reweighting with two-way constraints */
    /* step 1 inflation: */
    $\mathbf{y}^T = \exp(\beta \bar{\mathbf{x}} / \max \bar{\mathbf{x}})$
    /* step 2 bistochastic normalization : */
    repeat
        normalize across rows by $\mathbf{y}_{ij} = \mathbf{y}_{ij} / \sum_j \mathbf{y}_{ij}$
        normalize across columns by $\mathbf{y}_{ij} = \mathbf{y}_{ij} / \sum_i \mathbf{y}_{ij}$
    until $\mathbf{y}$ *converges*;
    $\mathbf{y} = \mathbf{y} / \sum \mathbf{y}_{ij}$
    /* Affinity-preserving random walking with reweighted jumps */
    $\mathbf{x}^T = \alpha \bar{\mathbf{x}}^T + (1 - \alpha) \mathbf{y}^T$
    $\mathbf{x} = \mathbf{x} / \sum \mathbf{x}_{ij}$
until $\mathbf{x}$ *converges*;
discretize $\mathbf{x}$ by the matching constraints

Appendix C

Lists

C.1 List of Figures

Appendix D

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*

Combinatorial 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, J. Lee, and K. M. Lee. Feature Correspondence and Deformable Object Matching via Agglomerative Correspondence Clustering. In *The IEEE International Conference on Computer Vision (ICCV)*, 2009.

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

[14] M. Cho and K. M. Lee. Progressive graph matching: Making a move of graphs via probabilistic voting. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12. IEEE Computer Society, 2012.

[15] M. Cho, J. Sun, O. Duchenne, and J. Ponce. Finding Matches in a Haystack : A Max-Pooling Strategy for Graph Matching in the Presence of Outliers. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[16] M. Cho, J. Sun, O. Duchenne, and J. Ponce. MATLAB code for Max-Pooling Graph Matching. http://www.di.ens.fr/willow/research/maxpoolingmatching/, 2014.

[17] 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.

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

[19] 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.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 3rd edition, 2009.

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

[22] 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.

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

[24] 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.

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

[26] M. Fiori, P. Sprechmann, J. Vogelstein, P. Muse, and G. Sapiro. Robust Multimodal Graph Matching: Sparse Coding Meets Graph Matching. *Advances in Neural Information Processing Systems*, (1):127–135, 2013.

[27] M. Fiori, P. Sprechmann, J. Vogelstein, P. Muse, and G. Sapiro. Robust Multimodal Graph Matching: Sparse Coding Meets Graph Matching, supplemental code. http://www.fing.edu.uy/~mfiori/, 2013.

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

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

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

[31] 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.

[32] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996. Available at http://books.google.de/books?id=mlOa7wPX6OYC, last access on 28/09/2015.

[33] P. E. Hart, N. J. Nillson, and B. Raphael. Formal Basis for the Heuristic Determination of Minimum Cost Path. *IEEE Transactions of Systems Science and Cybernetics*, 4(2):100–107, 1968.

[34] B. Hendrickson and R. Leland. A Multi-Level Algorithm For Partitioning Graphs. In *Proceedings of the IEEE/ACM SC95 Conference*, pages 1–14, 1995.

[35] 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.

[36] A. K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[37] G. Karypis and V. Kumar. Multilevel graph partitioning schemes. In *Proc. 24th Intern. Conf. Par. Proc., III*, pages 113–122. CRC Press, 1995.

[38] 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.

[39] 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.

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

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

[42] 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.

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

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

[45] 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.

[46] S. P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[47] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[48] Y. Lu, K. Huang, and C.-L. Liu. A fast projected fixed-point algorithm for large graph matching, 2012. Available at http://arxiv.org/abs/1207.1114v3, last access on 17/10/2015.

[49] 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.

[50] V. Lyzinski, D. L. Sussman, D. E. Fishkind, H. Pao, L. Chen, J. T. Vogelstein, Y. Park, and C. E. Priebe. Spectral Clustering for Divide-and-Conquer Graph Matching. 2011/14. Available at http://arxiv.org/abs/1310.1297v5, last access on 17/10/2015.

[51] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761 – 767, 2004. British Machine Vision Computing 2002.

[52] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981. Available at https://cs.anu.edu.au/people/Brendan.McKay/nauty/pgi.pdf, last access on 17/10/2015.

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

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

[55] A. Myronenko and X. Song. Point-Set Registration: Coherent Point Drift. *CoRR*, abs/0905.2635, 2009.

[56] 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.

[57] 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.

[58] H. Qiu and E. R. Hancock. Graph matching and clustering using spectral

partitions. *Pattern Recognition*, 39(1):22–34, 2006.

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

[60] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice Hall College Div, 1977.

[61] K. Rose. Determenistic annealing, clustering and optimization, 1991. Available at `http://resolver.caltech.edu/CaltechETD:etd-07122007-085228`, last access on 17/10/2015.

[62] 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.

[63] 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.

[64] 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.

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

[66] 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.

[67] C. Schellewald and C. Schnörr. Probabilistic subgraph matching based on convex relaxation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3757 LNCS:171–186, 2005.

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

[69] 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.

[70] K. Shearer, H. Bunke, and S. Venkatesh. Video sequence matching via

decision tree path following. *Pattern recognition letters*, pages 479–492, 2001.

[71] 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.

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

[73] 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.

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

[75] S. Umeyam. An Eigendecomposition Approach to Weighted Graph Matching Problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10, 1988.

[76] D. Van der Weken, M. Nachtegael, and E. Kerre. Some new similarity measures for histograms. In *ICVGIP*, pages 441–446, 2004.

[77] 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. Available at `http://arxiv.org/abs/1112.5507v5`, last access on 17/10/2015.

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

[79] 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.

[80] M. Zaslavskiy. Graph matching and its application in computer vision and bioinformatics, 2010.

[81] 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.

[82] M. Zaslavskiy, F. Bach, and J.-P. Vert. A path following algorithm for the

graph matching problem, GraphM library. http://cbio.ensmp.fr/graphm/, 2011.

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.

Place and date	Signature