Chapter 1

Graph Matching

In this chapter we review basic definitions and notations from the graph theory used in this thesis and introduce different forms and formulations of the graph matching problem together with some algorithms for solving them. The classification we use is based on [7], although we concentrated our attention mainly on one specific group of algorithms: those, which consider the graph matching problem as a quadratic optimization problem. Not all algorithms, we present thereby, were initially mentioned in [7], but we also do not cover all of the resent ones due to their quantity. We focus our selfs specially on those, that are important for further reading of the thesis.

1.1 Basic definitions and notation

A *undirected graph* $G = (V, E)$ is defined as a pair of disjoint sets V, E, where $E \subseteq \{\{u, v\} | u, v \in V\}$ [8]. The elements of the set V are called *vertices* or *nodes*[1] and the elements of E are called *edges*. Where it is necessary, we will write $V(G), E(G)$ to refer node and edge sets to the graph G.

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

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

and n is the number of nodes in the graph.

[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 $i = 0, \ldots, k$. 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. The subgraph G' of G is *induced by the set* $V' \subseteq V$, if $E' = \{(v_i, v_{i'})|v_i, v_{i'} \in V'\}$. Analog, the set $E' \subseteq E$ induces the subgraph G' of G, if $V' = \{v \in V|v \in e \text{ and } e \in E'\}$. For the induced subgraph we use the notation $G' = G[V']$ and $G' = G[E']$ [8], depending if it is node- or vertex-induced. We also define graph cut $G \cap G' = (V \cap V', E \cap E')$ and union $G \cup G' = (V \cup V', E \cup E')$.

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

Consider two undirected attributed graphs $G^I = (V^I, E^I, D^I)$ and $G^J = (V^J, E^J, D^J)$. We assume the situation, where $|V^I| = n_1$, $|V^J| = n_2$ and $n_1 \leq n_2$. A *matching function* between G^I and G^J is a mapping $m : V^I \to V^J$ 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 metric $S(G^I, G^J, m)$ to measure the quality of matching m. In this case, *Graph matching problem* between G^I and G^J can be defined as a problem of finding such a map $m : V^I \to V^J$, that maximizes the similarity score $S(G^I, G^J, m)$ between the graphs and fulfills some additional constraints:

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

Based on the required properties of the mapping m the algorithms, that solve this problem, can be divided into two large groups [7]: *exact* and *inexact* graph matching methods. There are also variations inside of each group based on the definition of the similarity score between two graphs. In the following sections we will give an overview of the common exact and inexact graph matching algorithms.

1.2 Exact graph matching

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

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

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

All problems except graph isomorphism are proofed to be $NP-$complete [9]. This can be shown through the reduction of the matching problem to the clique problem. The graph isomorphism is currently shown to be $NP-$hard. For special types of graphs there exist polynomial time algorithms (e.g. for trees [1, 9]).

The most widespread approach to solve exact graph matching problem is branch and bound [16] algorithm. It starts from some empty solution and tries in each step to expend it's current known solution based on some rules. If it happens, that a current solution does not fulfill problem's constraints, the solution is cut and algorithm backtracks to the last feasible solution and tries to expand it in other way. The most famous algorithm, that uses this principle is the one by Ullman [21]. Later it was extended and improved mostly by the suggestion of a new pruning heuristic (see for example [11] for comparison of algorithms based on the same technique).

The other techniques, that were successfully applied for the (sub)graph isomorphism

are decision trees [14, 20, 19] and group theory[13] for maximal subgraph matching.

1.3 Inexact graph matching

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

Let us recall the problem statement (1.1):

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

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

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

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

1.3.1 Graph Matching Objective Function

In some literature instead of defining the similarity function as in (1.1) one speaks about dissimilarity between two graphs cite or matching score cite. The goal in this case is to minimize those values. Two versions of the problem formulation are

however equivalent and one can easy transform one into other.

1.3.2 Quadratic Optimization Problem

The objective of graph matching is to find a mapping between two attributed graphs $G^I = (V^I, E^I, D^I)$ and $G^J = (V^J, E^J, D^J)$. Let the size of the graphs be n_1 and n_2 respectively. Generally, n_1 and n_2 can be different, but then we assume without losing generality $n_1 \leq n_2$. Here and further, we require the mapping m in (1.1) to be total and injective, which guarantees, that each node of the first graph will be matched to exactly one node of the second graph (one-to-one matching).

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

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

We denote with Π_n the set of all feasible permutations:

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

Let matrices A^I and A^J be the adjacency matrices of the graphs G^I and G^J respectively. We assume, that in case of weighed graphs, the adjacency matrices contain edge weights, instead of binary values. We can now formulate the graph matching problem as (compare with [12, 17, 22]cite):

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

where $\|\cdot\|$ is the matrix Frobenius norm. After some transformations, the problem (1.2) can be reformulated as (see [2], Appendix QAP?):

$$P = \underset{\hat{P} \in \Pi_n}{\operatorname{argmin}} vec(\hat{P})^T (-(A^J)^T \otimes (A^I)^T) vec(\hat{P}) \tag{1.3}$$

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

of $\hat{P} \in \Pi_n$.

The objective function of (1.3) is a negation of the objective of the *quadratic assignment problem* with linear cost matrix set to zero, which is know to be $NP-$hard [2, 18]. The both formulations have their advantages and disadvantages. The main benefit of (1.2) is the low space complexity $\mathcal{O}(n^2)$, where the space requirement of (1.3) estimates with $\mathcal{O}(n^4)$. This makes the second formulation tractable only for relative small graphs. The drawback of both formulations is the strict penalization function of edge disagreements, namely the squared euclidean distance between matched edges. This follows straight forward from (1.2). In this sense the last formulation can be easily generalized in the way, we represent below.

We return back to the case where $n_1 \neq n_2$. Let denote with a binary vector $x \in \{0,1\}^{n_1 n_2}$ the column-wise vectorization of the assignment matrix P, which is not necessary a permutation matrix anymore. It is obviously, that $x_{(j-1)n_1+i} = 1$, if node $v_i \in V^I$ is matched to 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 complicate $x_{(j-1)n_1+i}$.

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

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

$$S(G^I, G^J, m) = \sum_{\substack{x_{ij}=1 \\ x_{i'j'}=1}} s_E(e_{ij}, e_{i'j'}) + \sum_{x_{ij}=1} s_V(v_i, v_j) \tag{1.4}$$

This formula can be also expressed in matrix form. We define an *affinity* or *similarity matrix* $A \in \mathbb{R}^{n_1 n_2 \times n_1 n_2}$, whose diagonal elements are $s_V(v_i, u_j)$ and non-diagonal elements are $s_E(e_{ii\prime}, e_{jj\prime})$. Using this matrix we become the following formulation of

the graph matching problem as an quadratic optimization problem [3, 5, 6, 7]:

$$\underset{x}{\operatorname{argmax}}\, x^T S x \tag{1.5}$$

$$\text{s.t. } x \in \{0,1\}^{n_1 n_2} \tag{1.6}$$

$$\sum_{i=1\ldots n_1} x_{ij} \leq 1 \tag{1.7}$$

$$\sum_{j=1\ldots n_2} x_{ij} \leq 1 \tag{1.8}$$

We notice, that in the case, where $n_1 = n_2$, both conditions (1.7) and (1.8) will be fulfilled with equality.

One can easily see, that the formulation (1.3) can be obtained from (1.2)-(1.8) by setting $S = (A^J)^T \otimes (A^I)^T$.

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

- Node similarity

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

  $$s_V(v_i, v_j) = dist(d_i, d_j), \text{ for } \forall v_i \in V^I, \forall v_j \in V^J \tag{1.9}$$

  It can be, for example, an euclidean distance ([5]cite). The other widely used alternative to define node similarity, is to use cosine similarity[2] of node attributes [15]cite: $dist_c(d_i, d_j) = \frac{d_i \cdot d_j}{\|d_i\|_2 \|d_j\|_2}$, where dot signifies scalar product of two vectors.

- Edge similarity

  One possible approach to calculate edge similarity is to calculate *edge dissimilarity* $d_E : E^I \times E^J \to \mathbb{R}$ first and then transform it into similarity by using, for example, one of the following functions [3, 4, 5, 6]:

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

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

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

This leads us to the question how to calculate edge dissimilarity. The most obvious way is to compare a weights of the edges, if such are provided. An other alternative could be to use length of the edges [3, 4, 5, 6], if coordinates of the nodes in some system (usually, Cartesian coordinates) are known. It a significant assumptions, which holds however for almost all graphs arose in practical application.

If the nodes of the graphs is described not only by their location, but also by some affine region around the nodes, one can use so-called geometric dissimilarity [4, 6]:

$$d(e_{ij}, e_{i'j'}) = \frac{1}{2}(d_{geo}(m_j|m_i) + d_{geo}(m_i|m_j)) \tag{1.10}$$

$$d_{geo}(m_j|m_i) = \frac{1}{2}(\|x_{j'} - H_i x_j\| + \|x_j - H_i^{-1} x_{j'}\|) \tag{1.11}$$

$$d_{geo}(m_i|m_j) = \frac{1}{2}(\|x_{i'} - H_j x_i\| + \|x_i - H_j^{-1} x_{i'}\|) \tag{1.12}$$

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

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

1.3.3 Tree search

1.3.4 Quadratic Assignment Problem

1.3.5 Probabilistic approach

1.3.6 Spectral matching

1.3.7 Clustering matching

1.3.8 Reset algorithms for big graphs

1.4 Graph matching algorithms studied in this thesis

1.5 Discussion

Appendix A

Quadratic Assignment Problem

Consider a problem of assignment of n facilities to n locations given the transportation costs between the locations depending on the flow between them and opening costs of facilities in certain locations. The aim is to minimize the summary cost of the assignment. Let $D = (d_{kl}), F = (f_{kl}), B = (b_{ik}) \in \mathbb{R}^{n\times n}$ be a real matrices that define a distances ans flow between the locations, as well as opening costs. The problem define above can then be formulated as an integer quadratic program [2, 10]:

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

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

Here we want to show, who the formulation (A.1) can be transformed into (1.2) and (1.3). For that we omit first of all the opening costs B.

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

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

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

$$P = \underset{\hat{P}\in\Pi_n}{\operatorname{argmin}}(F \cdot XD^TX^T) \tag{A.2}$$

where $\cdot$ denotes the dot product.

$$P = \underset{\hat{P} \in \Pi_n}{\operatorname{argmin}} \operatorname{tr}(FXD^T X^T) \tag{A.3}$$

Appendix B

Bibliography

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[2] 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.

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

[4] 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.

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

[6] 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.

[7] 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.

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

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

[10] 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.

[11] 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.

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

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

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

[15] 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.

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

[17] 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.

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

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

[20] 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.

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

[22] 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.

Declaration of Authorship

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

Heidelberg, den (Datum) .....