UNIVERSITY OF MANNHEIM

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Analysis of a Deterministic Annealing Method for Graph Matching and Quadratic Assignment Problems in Computer Vision

DIPLOMA THESIS

by
Stefan Roth

Computer Vision, Graphics, and Pattern Recognition Group

Mai 2001

Thesis Supervisor: Prof. Dr. Christoph Schnörr
Co-Referee: Prof. Dr. Matthias Krause

Odi et amo. Quare id faciam, fortasse requiris.
Nescio, sed fieri sentio et excrucior.

Catullus, Carmen 85

Preface

This thesis was written in partial fulfillment of the requirements for attaining the *Diploma* degree in "Technische Informatik" (Computer Science and Engineering) at University of Mannheim, Mannheim, Germany.

It is the result of research conducted at the Computer Vision, Graphics, and Pattern Recognition Group, which is part of the Department of Mathematics and Computer Science at University of Mannheim. The work was supervised by Prof. Dr. Christoph Schnörr, whom I wish to thank very much for giving me the opportunity to work in the stimulating environment of his group as well as for his support during the creation of this work.

I am indebted to Christian Schellewald, Daniel Cremers, Dr. Joachim Weickert, and all other people of the CVGPR-Group for their valuable suggestions and discussions as well as for the pleasant working atmosphere. I am much obliged to Florian Tischhäuser for his patience and helpful contributions while he shared the office with me. Special thanks also goes to Matthias Heiler, who particularly assisted me with English grammar.

Finally, I wish to thank my parents Klaus-Dieter and Waltraud Roth for their continuous support during my studies and during the creation of this thesis.

Abstract

View-based object recognition is a central area of computer vision research. In many approaches this amounts to solving graph matching problems. This thesis analyzes the *soft-assign quadratic assignment algorithm*, a recent deterministic annealing method for approximately solving graph matching and quadratic assignment problems proposed by Gold and Rangarajan [7]. Since both problems are NP-hard, approximation algorithms with polynomial running time are very important. After a step-by-step derivation of the algorithm, the first part of this work analyzes the soft-assign quadratic assignment algorithm theoretically regarding parameter choice and convergence related properties. Furthermore, the applicability to special problem classes is studied. The relation to other neural network techniques is described as well. The second part of this thesis is devoted to an experimental analysis of the algorithm using several thousand graph matching and quadratic assignment problems of various type. Finally, the robustness of the algorithm with respect to changes of the algorithm parameters is studied.

Categories and subject descriptors: G.2.2 Graph theory — Graph algorithms; G.2.1 Combinatorics — Combinatorial algorithms; I.4.8 Scene analysis — Object recognition

Keywords: graph matching, weighted graphs, quadratic assignment problem, deterministic annealing, soft-assign, feature matching

Zusammenfassung

Ansichten-basierte Objekterkennung ist ein zentrales Gebiet der Forschung im Bereich Maschinensehen. In vielen Ansätzen ist diese äquivalent zum Lösen von Graph-Matching Problemen. Die vorliegende Diplomarbeit untersucht den *Soft-Assign Quadratic Assignment* Algorithmus, einen deterministischen Annealing Ansatz, um Graph-Matching und Quadratic-Assignment Probleme näherungsweise zu lösen. Dieser wurde von Gold und Rangarajan [7] vorgeschlagen. Da beide Probleme NP-schwer sind, kommt Näherungsalgorithmen, die eine polynomielle Laufzeit besitzen, eine besondere Bedeutung bei. Nach einer schrittweisen Herleitung dieses Verfahrens wird im ersten Teil dieser Arbeit der Soft-Assign Quadratic Assignment Algorithmus theoretisch hinsichtlich Parameterwahl und Konvergenzeigenschaften untersucht. Weiterhin wird die Anwendbarkeit auf spezielle Problemklassen erläutert. Die Verbindung zu Neuronalen Netzen wird ebenfalls beschrieben. Der zweite Teil dieser Diplomarbeit ist der experimentellen Analyse dieses Algorithmus mittels tausender verschiedener Graph-Matching und Quadratic-Assignment Probleme gewidmet. Die Robustheit des Algorithmus bezüglich der Änderung seiner Parameter wird ebenfalls untersucht.

Notation

The following section summarizes the notational conventions used throughout this thesis. It is complemented by table 0.1, which sums up important symbols used.

Vectors are typeset in bold lower case letters, e. g., $\boldsymbol{x}$. The elements of a vector are denoted as a non-bold lower case letters, i. e., the i-th element of vector $\boldsymbol{x}$ is noted as x_i.

Matrices are represented by bold capital letters, e. g., $\boldsymbol{A}$. The elements of a matrix are typeset in non-bold capital letters using a row-major order for indexing, i. e., A_{ij} represents the element in the i-th row and the j-th column of matrix $\boldsymbol{A}$. Arrays are noted in a similar fashion; in case of more than two dimensions only the number of indices is changed. Occasionally, indices may be grouped using a semicolon, e. g., $\boldsymbol{A}_{ij;kl}$ represents a 4-dimensional array.

If certain symbols are used within an iteration scheme, the iteration number is sometimes clarified with a superscript, i. e., $\boldsymbol{M}^{(n)}$ represents matrix $\boldsymbol{M}$ at iteration step n.

Symbol	*Meaning*
$\boldsymbol{x}^T, \quad \boldsymbol{A}^T$	Transposed vector respective matrix
$\mathrm{tr}\{\boldsymbol{A}\}$	Trace of matrix $\boldsymbol{A}$
$\Vert\cdot\Vert_{\mathrm{F}}$	Frobenius norm of a matrix
$\otimes$	Kronecker product of two matrices (see also A.2)
$\boldsymbol{I}_n$	Identity matrix of size $n \times n$
$\mathbf{1}_{n\times m}$	Matrix of ones of size $n \times m$
$\mathbf{0}_{n\times m}$	Zero matrix of size $n \times m$
$\mathrm{diag}\{\boldsymbol{x}\}$	Diagonal matrix constructed using the elements of vector $\boldsymbol{x}$
$\mathrm{vec}(\boldsymbol{A})$	Vector obtained by stacking the columns of a matrix $\boldsymbol{A}$
$\lambda_{\max}(\boldsymbol{A}), \lambda_{\min}(\boldsymbol{A})$	Maximal and minimal eigenvalue of matrix $\boldsymbol{A}$; the argument may be skipped in a non-ambiguous context
Π_n	Set of all permutations
$\hat{\Pi}_n$	Set of all permutation matrices of size $n \times n$
$\tilde{\Pi}_{n,m}$	Set of all match matrices of size $n \times m$

Table 0.1: Further notational conventions

Symbol	*Meaning*
$\boldsymbol{P}$	Permutation matrix
$\boldsymbol{M}$	Assignment matrix (doubly stochastic)
$\boldsymbol{C}$	Generalized quadratic cost matrix of a QAP
$\boldsymbol{B}$	Linear cost matrix of a QAP

Table 0.1: Further notational conventions

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Introduction and Overview

Visual recognition of objects is one of the central problems in computer vision research. Several methods of representing objects for recognition in computer vision systems have been proposed; among others, *view-based* object representations gained considerable interest. One common way of representing object views for recognition is a set of image features with relations on them.

Local image features may, for example, include points, line segments, or curve segments. A powerful way of describing links between features are pairwise relations (in a mathematical sense), which have a real value attached that encodes, e. g., the similarity of the features or their spatial proximity. A set along with pairwise relations can be represented as a *simple*[1]*, weighted, undirected graph*. Figure 1.1 gives an example of such a view-based graph (without showing the similarity measures). Thus, recognizing objects by comparing features and relations obtained from two views can be reduced to comparing two corresponding graphs. Comparing graphs is better known as *inexact graph matching*[2], i. e., finding the correspondence between the nodes of the graphs that maximizes their similarity.

[1]A graph without self-loops.

[2]Inexact matching refers to matching of non-isomorphic graphs. It should not be confused with approximately solving matching problems.

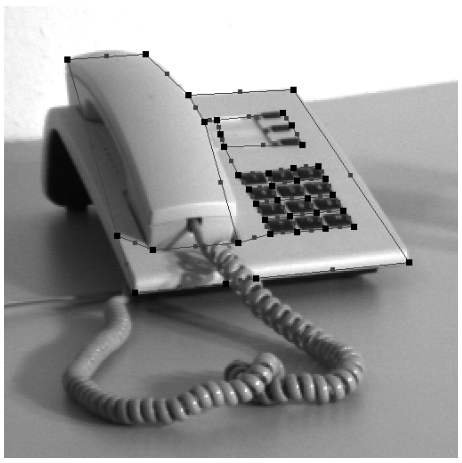


Figure 1.1: A hand designed graph using automatically detected features with 38 nodes. The features were detected using the publicly available feature extraction system FEX [1].

Unfortunately, graph matching is a hard combinatorial optimization problem; in particular, it is NP-hard. This makes solving matching problems with medium-sized graphs of, e. g., 20 nodes intractable on today's computers. Therefore, there is a strong interest for approximation algorithms that find good suboptimal solutions and have a low-order polynomial time complexity. A graph matching algorithm that gained particular attention through its excellent experimental performance is the *soft-assign quadratic assignment algorithm* proposed by Rangarajan et al. [24]. This thesis provides an analysis of the theoretical and experimental properties of this algorithm. Theoretical aspects include the evaluation of convergence related characteristics and the estimation of algorithm parameters based on theoretical insights. This is complemented by assessing the performance of the algorithm for various problems and the robustness of the algorithm with respect to parameter changes. The reader should note that this thesis is not concerned with the techniques for obtaining graphs from object views.

In the remainder of this chapter, we will formally introduce the graph matching problem and later show it to be a special case of the *quadratic assignment problem*, a classical problem in combinatorial optimization. Following that, we will formally introduce quadratic assignment problems, or shortly QAPs, supplemented by several important formulations and prominent examples. Finally, a brief summary will be given about important approximation algorithms for QAPs and graph matching problems.

Chapter 2 begins with a step-by-step derivation of the soft-assign quadratic assignment algorithm. Furthermore, we will discuss special problem cases, followed by theoretical properties regarding convergence of the algorithm and parameter choice. In chapter 3 we shall see the relation of the soft-assign quadratic assignment algorithm to other deterministic annealing techniques. In chapter 4, several thousand experiments conducted using this algorithm on various graph matching and quadratic assignment problems are documented. A further study of the robustness regarding the algorithm parameters is followed by a discussion of the findings of all experiments. The terminal chapter will give a summary of the results and observations.

1.2 Graph Matching and Quadratic Assignment Problems

1.2.1 Problem definition

Before we define the graph matching problem itself, let us first define the graphs to be matched. For reasons of simplicity, only simple, weighted, undirected graphs are considered here[3]. In the following, we assume two graphs to be given, denoted as $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Furthermore, both graphs are assumed to have an equal number of vertices $|V_1| = |V_2| = n$. The case of graphs with unequal numbers of vertices will be discussed in section 2.4.1. V_1, V_2 are indexable sets of *vertices* or *nodes*[4] $\{v_{p;0}, \ldots, v_{p;n}\}$, $p = 1, 2$. The *edges* of the respective graph are denoted as $E_1 \subseteq V_1 \times V_1$ and $E_2 \subseteq V_2 \times V_2$. Each graph has an associated *weight function*, which assigns a positive real weight[5] to every edge ($w_1 : E_1 \to \mathbb{R}_+$ and $w_2 : E_2 \to \mathbb{R}_+$). We can deduct corresponding *adjacency matrices* from the weight functions:

$$\mathbb{R}^{n\times n} \ni \boldsymbol{A}_p := \left(A_{p;ij}\right)_{i,j=1}^{n}, \quad A_{p;ij} := \begin{cases} w_p\left((v_{p;i}, v_{p;j})\right), & (v_{p;i}, v_{p;j}) \in E_p \\ 0, & \text{otherwise} \end{cases}, \quad p = 1, 2 \tag{1.1}$$

$\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are symmetric, non-negative matrices, since the graphs are undirected and the weight function is assumed to be positive. For the special case of unweighted graphs, the adjacency matrices are confined to the domain of binary matrices.

The matching itself is a bijective mapping between the two sets of vertices V_1 and V_2. For integer sets $\{1, \ldots, n\}$ such a bijective mapping is better known as *permutation*. We define the set of all permutations as usual:

$$\Pi_n := \left\{\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\} \;\middle|\; \sigma \text{ bijective } \right\} \tag{1.2}$$

It is convenient for us to conceive the matching as permutation between the indices of the elements of the sets. In the remainder of this work, the term matching usually refers to a permutation in this sense.

For a permutation σ we define a *permutation matrix* $\boldsymbol{P}$, which permutes the components of a vector according to σ:

$$\mathbb{R}^{n\times n} \ni \boldsymbol{P} = \left(P_{ij}\right)_{i,j=1}^{n}, \quad P_{ij} := \begin{cases} 1, & \sigma(i) = j \\ 0, & \text{otherwise} \end{cases} \tag{1.3}$$

We can easily verify this as follows:

$$(\boldsymbol{Px})_i = \sum_{j=1}^{n} P_{ij} x_j = x_{\sigma(i)}, \qquad \boldsymbol{x} \in \mathbb{R}^n \tag{1.4}$$

Permutation matrices may also be used for permuting rows and columns of a matrix:

$$\left(\boldsymbol{PAP}^{\mathrm{T}}\right)_{ij} = \sum_{k,l=1}^{n} P_{ik} A_{kl} P_{jl} = A_{\sigma(i)\sigma(j)}, \qquad \boldsymbol{A} \in \mathbb{R}^{n\times n} \tag{1.5}$$

[3] Attributed relational graphs are common as well, but the definitions and algorithms in this work can be extended to them in a quite straightforward way. Section A.1 will give a brief summary of these extensions.

[4]Node and vertex are used as synonyms throughout this work.

[5]In case of view-based object recognition this weight comprises a similarity measure.

This property is of particular importance here, because the adjacency matrices of graphs can, of course, be permuted using them. Later, we will also use matrices for which only the columns are permuted. This is achieved by

$$\left(\boldsymbol{A}\boldsymbol{P}^{\mathrm{T}}\right)_{ij} = \sum_{k=1}^{n} A_{ik}P_{jk} = A_{i\sigma(j)}, \qquad \boldsymbol{A} \in \mathbb{R}^{n\times n}. \tag{1.6}$$

From the definition of permutation matrices (1.3) and the bijectivity property of permutations, we observe that permutation matrices are binary matrices with rows and columns each summing to one[6]. We use this fact to define the set of all permutation matrices as

$$\hat{\Pi}_n := \left\{ \boldsymbol{P} \in \{0,1\}^{n\times n} \,\middle|\, \forall j: \sum_{i=1}^{n} P_{ij} = 1,\ \forall i: \sum_{j=1}^{n} P_{ij} = 1 \right\}. \tag{1.7}$$

We can observe that the set of permutation matrices is governed by three different types of constraints: an integrality constraint, row sum constraints, and column sum constraints. For the time being, optimizing over the domain of permutations and optimizing over the domain of permutation matrices is regarded as equivalent, since they are easily transformed into one another.

Let us return to graph matching. The objective of graph matching is to permute one of the graphs such that the similarity of the permuted graph and its counterpart is maximized. The similarity of the graphs is usually expressed by the conformity of their edges and edge weights, or equivalently by the conformity of their adjacency matrices. One way to express this formally is to minimize the difference between the two adjacency matrices, one of which is permuted, measured by some (squared) matrix norm. For simplicity, we use the Frobenius matrix norm resulting in the following definition of *weighted graph matching*:

$$\boldsymbol{P}^* = \arg\min_{\boldsymbol{P}\in\hat{\Pi}_n} \|\boldsymbol{P}\boldsymbol{A}_1\boldsymbol{P}^{\mathrm{T}} - \boldsymbol{A}_2\|_{\mathrm{F}}^2 \tag{1.8}$$

After we replace the matrix norm and perform some further simplifications, we obtain the more intuitive objective

$$\sigma^* = \arg\min_{\sigma\in\Pi_n} \sum_{i,j=1}^{n} \left(A_{1;\sigma(i)\sigma(j)} - A_{2;ij}\right)^2. \tag{1.9}$$

We can transform the objective function (1.8) into a more manageable form:

$$\begin{aligned}
\|\boldsymbol{P}\boldsymbol{A}_1\boldsymbol{P}^{\mathrm{T}} - \boldsymbol{A}_2\|_{\mathrm{F}}^2 &= \operatorname{tr}\left\{(\boldsymbol{P}\boldsymbol{A}_1\boldsymbol{P}^{\mathrm{T}} - \boldsymbol{A}_2)(\boldsymbol{P}\boldsymbol{A}_1\boldsymbol{P}^{\mathrm{T}} - \boldsymbol{A}_2)^{\mathrm{T}}\right\} \\
&= \operatorname{tr}\left\{(\boldsymbol{P}\boldsymbol{A}_1\boldsymbol{P}^{\mathrm{T}} - \boldsymbol{A}_2)(\boldsymbol{P}\boldsymbol{A}_1^{\mathrm{T}}\boldsymbol{P}^{\mathrm{T}} - \boldsymbol{A}_2^{\mathrm{T}})\right\} \\
&= \operatorname{tr}\left\{\boldsymbol{P}\boldsymbol{A}_1\boldsymbol{P}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{A}_1^{\mathrm{T}}\boldsymbol{P}^{\mathrm{T}} - \boldsymbol{P}\boldsymbol{A}_1\boldsymbol{P}^{\mathrm{T}}\boldsymbol{A}_2^{\mathrm{T}} - \boldsymbol{A}_2\boldsymbol{P}\boldsymbol{A}_1^{\mathrm{T}}\boldsymbol{P}^{\mathrm{T}} + \boldsymbol{A}_2\boldsymbol{A}_2^{\mathrm{T}}\right\} \\
&= \operatorname{tr}\left\{\boldsymbol{A}_1^{\mathrm{T}}\boldsymbol{A}_1 - 2\boldsymbol{A}_2\boldsymbol{P}\boldsymbol{A}_1^{\mathrm{T}}\boldsymbol{P}^{\mathrm{T}} + \boldsymbol{A}_2\boldsymbol{A}_2^{\mathrm{T}}\right\}
\end{aligned} \tag{1.10}$$

[6] It is easily verified that this property is not only necessary, but also sufficient.

Because constant summands and constant positive factors may be ignored when minimizing a function, the equivalence

$$\arg\min_{P \in \hat{\Pi}_n} \|PA_1P^T - A_2\|_F^2 = \arg\min_{P \in \hat{\Pi}_n} -\operatorname{tr}\left\{A_2PA_1^TP^T\right\} \tag{1.11}$$

immediately follows. The latter optimization problem has the form of a *quadratic assignment problem* (without linear term). QAPs are of central importance in computer science, because many hard combinatorial optimization problems belong to this class. In the following section we will discuss quadratic assignment problems, their definition, and their applications, because the graph matching algorithm analyzed in this work can easily be generalized to quadratic assignment problems.

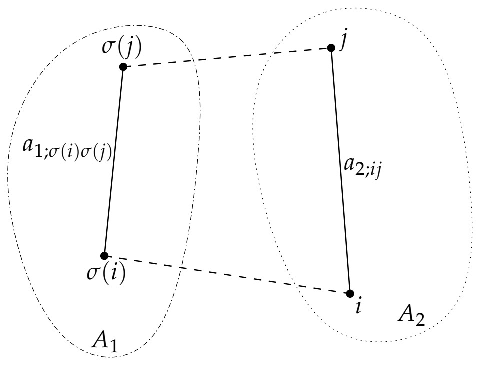


Figure 1.2: Correspondence between two matched node pairs and their edges

1.2.2 Quadratic assignment problems

Quadratic assignment problems are defined in various ways throughout literature, often depending on the specific application. One major difference between the formulations is the consideration of a linear term in addition to the quadratic term. The quadratic cost also bears some differences; some variants use two cost matrices, others a single four-dimensional array. In the following paragraphs we will introduce the QAP in its original form and also mention some of the most commonly used formulations. This section is only concerned with quadratic assignment problems between sets of equal size, since this is the most commonly used type. We will discuss problems with sets of different cardinality in section 2.4.1.

Quadratic assignment problems were introduced in 1957 by Koopmans and Beckman [14], motivated by the following economical problem: A set of n facilities is to be assigned to a set of n locations with given cost for the distances between the facilities and with the flow between them, plus additional cost for assigning a facility to a location. The costs are given by three, real $n \times n$ matrices F, D, and B, where F_{ij} is the flow between facility i and facility j, D_{ij} is the distance between location i and location j, and B_{ij} is the cost for assigning facility i to location j. The objective is to minimize the overall cost, what we can formally express by

$$\sigma^* = \arg\min_{\sigma \in \Pi_n} \sum_{i,j,k,l=1}^{n} F_{ij} D_{\sigma(i)\sigma(j)} + \sum_{i=1}^{n} B_{i\sigma(i)}. \tag{1.12}$$

Lawler [16] introduced a slightly generalized form of the quadratic assignment problem by replacing the cost matrices $\boldsymbol{F}$ and $\boldsymbol{D}$ by a single, four-dimensional array $\boldsymbol{C} = \left(C_{ij;kl}\right)_{i,j,k,l=1}^{n}$. The generalized problem is defined as

$$\sigma^* = \arg\min_{\sigma \in \Pi_n} \sum_{i,j,k,l=1}^{n} C_{ij;\sigma(i)\sigma(j)} + \sum_{i=1}^{n} B_{i\sigma(i)}. \tag{1.13}$$

Other formulations of quadratic assignment problems

Over the time, other formulations for QAPs evolved out of the two objectives above. One of the reformulations, which we shall study below, is useful for clarifying the relation between graph matching and quadratic assignment problems. Another will later be helpful to understand the similarity between the four-dimensional cost array of the Lawler form and a cost matrix, which allows to determine the eigenvalue spectrum of the costs.

The first reformulation is based on the original Koopmans-Beckmann form. Using the identity $\sum_{i,j=1}^{n} A_{ij}B_{ij} = \sum_{i,j=1}^{n} (\boldsymbol{A}\boldsymbol{B}^{\mathrm{T}})_{ii} = \operatorname{tr}\left\{\boldsymbol{A}\boldsymbol{B}^{\mathrm{T}}\right\}$ and the above mentioned properties of permutation matrices, we obtain the trace formulation

$$\boldsymbol{P}^* = \arg\min_{\boldsymbol{P} \in \hat{\Pi}_n} \operatorname{tr}\left\{\boldsymbol{F}\boldsymbol{P}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{P}^{\mathrm{T}} + \boldsymbol{B}\boldsymbol{P}^{\mathrm{T}}\right\}. \tag{1.14}$$

By comparing equations (1.14) and (1.11), we observe that graph matching is a special case of the quadratic assignment problem. Moreover, if the definition of graph matching would be extended to directed graphs with possible self-loops and possible negative edge weights, large parts of the QAPs could be conceived as graph matching problems, namely those without linear costs.

For obtaining the finally considered formulations, we apply some of the properties of Kronecker products, which are given in section A.2, to the trace formulation:

$$\begin{aligned} \operatorname{tr}\left\{\boldsymbol{F}\boldsymbol{P}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{P}^{\mathrm{T}}\right\} &= \operatorname{tr}\left\{\boldsymbol{P}^{\mathrm{T}}\boldsymbol{F}\boldsymbol{P}\boldsymbol{D}^{\mathrm{T}}\right\} = (\operatorname{vec}(\boldsymbol{P}))^{\mathrm{T}} \operatorname{vec}\left(\boldsymbol{F}\boldsymbol{P}\boldsymbol{D}^{\mathrm{T}}\right) \\ &= (\operatorname{vec}(\boldsymbol{P}))^{\mathrm{T}} (\boldsymbol{D} \otimes \boldsymbol{F}) \operatorname{vec}(\boldsymbol{P}) \end{aligned} \tag{1.15}$$

This immediately leads to the Kronecker formulation

$$\boldsymbol{P}^* = \arg\min_{\boldsymbol{P} \in \hat{\Pi}_n} (\operatorname{vec}(\boldsymbol{P}))^{\mathrm{T}} (\boldsymbol{D} \otimes \boldsymbol{F}) \operatorname{vec}(\boldsymbol{P}) + (\operatorname{vec}(\boldsymbol{B}))^{\mathrm{T}} \operatorname{vec}(\boldsymbol{P}). \tag{1.16}$$

We can formulate the generalized form of the QAP by replacing the Kronecker product with an arbitrary real matrix $\boldsymbol{C}$:

$$\boldsymbol{P}^* = \arg\min_{\boldsymbol{P} \in \hat{\Pi}_n} (\operatorname{vec}(\boldsymbol{P}))^{\mathrm{T}} \boldsymbol{C} \operatorname{vec}(\boldsymbol{P}) + (\operatorname{vec}(\boldsymbol{B}))^{\mathrm{T}} \operatorname{vec}(\boldsymbol{P}) \tag{1.17}$$

For the time being, we do not strictly distinct between the matrix $\boldsymbol{C}$ and the four-dimensional cost array $\left(C_{ij;kl}\right)_{i,j,k,l=1}^{n}$ of the Lawler form. These two forms can be associated with one another by defining $\boldsymbol{C} := \left(\left(C_{ij;kl}\right)_{k,l=1}^{n}\right)_{i,j=1}^{n}$, which is a matrix of submatrices.

1.2.3 Other graph matching objectives

Gold and Rangarajan [7] proposed an alternative graph matching objective function, which directly defines the graph matching problem as quadratic assignment problem. As we shall see in section 2.4.1, this alternative objective function will be needed for matching graphs with unequal number of nodes with the soft-assign quadratic assignment algorithm.

We have seen that the traditional graph matching objective can be conceived as a quadratic assignment problem. In the generalized Lawler form the quadratic cost amounts to

$$C_{ij;kl} = -A_{2;ij}A_{1;kl}\,, \tag{1.18}$$

assuming that the two graphs have the adjacency matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$. The linear cost matrix $\boldsymbol{B}$ is set to zero as for all weighted graph matching problems.

Gold and Rangarajan [7] generalized the traditional approach toward an arbitrary edge weight compatibility function. They defined the quadratic cost as

$$C_{ij;kl} := \begin{cases} 0, & A_{2;ij} = 0 \ \vee \ A_{1;kl} = 0 \\ -f(A_{1;kl}, A_{2;ij}), & \text{otherwise,} \end{cases} \tag{1.19}$$

where $f(\cdot,\cdot)$ is the edge weight compatibility function. $C_{ij:kl}$ is explicitly defined to be 0, if at least one of the vertex pairs is not connected, because this ensures that the cost matrix $\boldsymbol{C}$ is sparse, if the graphs are sparse. The sparsity can be exploited in graph matching algorithms (cf. section 2.2.2). We should note here that the compatibility function f is not invariant to constant summands, because missing links always have zero cost.

1.2.4 Complexity

Before we discuss the complexity of the aforementioned problems, let us briefly review their relationship. Intuitively, the unweighted graph matching problem is a special case of the weighted graph matching problem. Furthermore, we showed that weighted graph matching problems are special cases of quadratic assignment problems.

A problem that is closely related to unweighted graph matching, is the subgraph-isomorphism problem. It is asking, whether a graph G_1 is a subgraph of another graph G_2. If the optimal matching of these two graphs is known, we can efficiently (i. e., in polynomial time) verify, whether G_1 is a subgraph of G_2. Since the subgraph-isomorphism problem is known to be NP-complete [see 5, p. 960], unweighted as well as weighted graph matching and QAPs are all *NP-hard*. Hence, it is unlikely that solutions to these problems can be efficiently computed in the sense of a polynomial time complexity. This bears many practical problems, since the computation of an exact solution, even for modest size problems, requires a huge amount of time. This is a result of the enormous combinatorial search space of these problems. For instance, if $n = 20$, there exist more than 10^{18} possible assignments. As an example, exactly solving the problem Nug17 from the QAPLIB library [3] with $n = 17$ required about 1.5 hours on a current PC (Pentium III, 700 MHz) using a branch-and-bound implementation distributed with the QAPLIB data. The QAPLIB is a collection of hard quadratic assignment problems from various sources, which are commonly used as benchmark experiments.

It has, however, been shown that there are a few special cases, in which QAPs can be solved in polynomial time (Pardalos et al. [21, p. 6], Burkard et al. [2, sec. 10]). These articles also point out that QAPs belong to the strongly NP-hard problems, because finding an ϵ-approximate[7] solution is also NP-hard.

Thus, the probably only way to alleviate the time requirements for the general case is confining to good, suboptimal solutions. Section 1.4 will discuss some of the various suboptimal approaches, which have been tried.

1.3 Some Example Quadratic Assignment Problems

In this section we give further examples of quadratic assignment problems, namely the *traveling salesman problem* (TSP) and the *graph partitioning problem* (GPP). Even though both are not directly related to computer vision applications, they are well studied, classical problems. We discuss them here in order to point out the breadth of the class of quadratic assignment problems. Naturally, specialized algorithms for these problems are superior to all known QAP algorithms, since specialized methods can exploit the structure of the problem [2, sec. 13.4].

1.3.1 Traveling salesman problem

As the name suggests, this problem is illustrated with a salesman, who has to visit n cities. The cities are represented as vertices of an undirected, complete graph, whose weights encode the distances between the cities. The problem is to find the shortest tour, i. e., an order, in which the salesperson travels through every city without visiting a city more than once. The traveling salesman problem is known to be NP-hard [see 5, pp. 959f].

As [21, p. 12] points out, the TSP can be formulated as QAP by taking the distance matrix of the TSP and an adjacency matrix of an arbitrary tour as distance and flow matrices of a QAP. We can, for example, define the latter cost matrix as

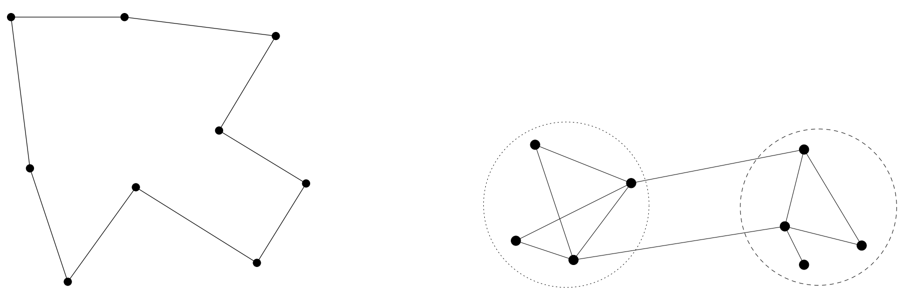
$$\mathbb{R}^{n\times n} \ni \boldsymbol{F} = \left(F_{ij}\right)_{i,j=1}^{n}, \qquad F_{ij} = \begin{cases} 1, & (i+1=j) \bmod n \\ 0, & \text{otherwise.} \end{cases} \tag{1.20}$$

We can verify that this is a valid tour through all cities. By permuting the adjacency matrix $\boldsymbol{F}$ of the tour, we can obtain any possible sequence of cities and hence any valid tour. Thus, minimizing the QAP objective

$$\boldsymbol{P}^* = \arg\min_{\boldsymbol{P}\in\hat{\Pi}_n} \operatorname{tr}\left\{\boldsymbol{FPD}^{\mathrm{T}}\boldsymbol{P}^{\mathrm{T}}\right\} \tag{1.21}$$

finds the shortest tour.

[7] ϵ-approximate solutions are characterized by a fixed maximum deviation from the optimum.



(a) TSP: A tour through 9 cities

(b) Graph bisection problem

Figure 1.3: Some example quadratic assignment problems

1.3.2 Graph partitioning

Graph partitioning is the problem of dividing the set of nodes of an arbitrary graph into $k \geq 2$ equal-sized subsets, while minimizing the flow between the partitions. If $k = 2$, the problem is also called graph bisection problem. For every $k \geq 2$ this problem is known to be NP-hard.

The graph bisection problem is mapped onto a QAP by taking the adjacency matrix of the graph as flow matrix; the distance matrix is a combined adjacency matrix of two disjoint, complete graphs with $\frac{n}{2}$ nodes each (Pardalos et al. [21, p. 12], Burkard et al. [2, sec. 13.4]). We may, for example, define the latter as

$$\mathbb{R}^{n \times n} \ni \boldsymbol{D} := \begin{pmatrix} \mathbf{1}_{\frac{n}{2} \times \frac{n}{2}} & \mathbf{0}_{\frac{n}{2} \times \frac{n}{2}} \\ \mathbf{0}_{\frac{n}{2} \times \frac{n}{2}} & \mathbf{1}_{\frac{n}{2} \times \frac{n}{2}} \end{pmatrix}.$$

To match our definition of QAPs, we have to reverse the sign of one of the cost matrices amounting to the QAP

$$\boldsymbol{P}^* = \arg \min_{\boldsymbol{P} \in \hat{\Pi}_n} \operatorname{tr} \left\{ (-\boldsymbol{F}) \boldsymbol{P} \boldsymbol{D}^{\mathrm{T}} \boldsymbol{P}^{\mathrm{T}} \right\}. \tag{1.22}$$

This maximizes the flow within the partitions of the graph and hence minimizes the flow between them.

We can also generalize the preceding approach to more than 2 subsets. For that, the distance matrix is defined as the combined adjacency matrix of k disjoint, complete graphs with $\frac{n}{k}$ nodes each. The flow matrix is again given by the adjacency matrix of the graph. The remark regarding the sign reversal applies here as well.

1.4 Well-known Approximation Algorithms

1.4.1 Discrete optimization methods

Construction methods

Construction methods[8] are among the earliest types of approximation algorithms for QAPs. They employ the step-by-step construction of partial assignments starting with an empty assignment. At each step, a certain assignment between a not yet considered facility and a not yet assigned location is added to the partial assignment, until a permutation is reached. The assignment is selected using a local heuristics, for example by considering all possible 2-way exchanges of an initial, user-supplied permutation. A 2-way exchange is a reversal of the mapping between two elements in one set and their assigned counterparts in the other set. The latter heuristics is used by a variant of the *CRAFT* algorithm, one of the oldest heuristics for quadratic assignment problems.

Improvement methods

Improvement methods[9] start with a feasible solution and try to improve it afterward. The improvement is achieved by local search algorithms, which are usually governed by the type of neighborhood considered and by the order of neighborhood traversal. For example, one might consider all 2-way or cyclic 3-way exchanges in a random order.

A widely used local search algorithm is *tabu search*. The tabu search tries to circumvent poor local minima by considering only a part of the neighborhood, in particular only neighbors that are not on the current tabu list. The tabu list is iteratively updated, whereby new tabu rules are added and others are removed. Various types of tabu search algorithms have been employed for solving QAPs approximately [cf. 2, sec. 8.4]. Certain types showed the highest performance for some of the problems in the QAPLIB.

Simulated annealing

Simulated annealing methods[10] are basically an extension of improvement methods. They are able to overcome certain local minima by occasionally allowing deteriorating changes. They belong to the group of *Monte-Carlo* algorithms. The probability for these deteriorating changes is gradually lowered according to an annealing schedule. The term "annealing" roots in a similarity to models of thermal annealing processes in statistical mechanics. Simulated annealing methods have, under mild conditions, been shown to converge toward the global optimum. Theoretical results regarding the convergence speed, however, are lacking until today [2, sec. 8.5]. In practice, these methods tend to be slow, but turned out to be most competitive for certain QAPLIB problems.

[8] Pardalos et al. [21, p. 23], Burkard et al. [2, sec. 8.1]
[9] Pardalos et al. [21, pp. 24f], Burkard et al. [2, sec. 8.3f]
[10] Pardalos et al. [21, p. 25f], Burkard et al. [2, sec. 8.5]

1.4.2 Continuous optimization methods

Relaxations

Relaxation based methods, an important class of techniques for QAPs, have in common that they drop binary constraints forming a continuous optimization problem, which may, e. g., be linear or convex quadratic [21, pp. 4f]. The purpose of these relaxations is to obtain a problem, whose global optimization is tractable in practice. After finding the global optimum of the relaxed problem, the obtained solution has to be transformed back to the domain of permutation matrices.

Furthermore, these techniques have the advantage that they can not only be used to find approximate solutions, but also to create branch-and-bound algorithms for computing the exact solution [2, sec. 6]. This bases on the fact that relaxations deliver a lower bound to the minimum of the original problem. The lower bound can be used to cut branches off the combinatorial search tree that do not lead to the global optimum. Nevertheless, such search algorithms naturally have exponential worst-case time requirements.

Deterministic annealing

The last class of algorithms, which we will introduce here, are algorithms using the deterministic annealing technique. Like the simulated annealing methods discussed above, these algorithms use an annealing schedule in analogy to physical annealing processes. In contrast to simulated annealing, deterministic annealing uses, as the name suggests, deterministic subroutines at every computational temperature on the schedule.

Prominent algorithms of this class use the annealing parameter to control the "degree of convexity" of a modified objective function. Usually, a convex term, which is weighted with the annealing parameter, is added to the objective function on the continuous domain. At high computational temperatures the overall objective function is convex and hence easy to minimize. By gradually lowering the temperature, local minima appear, which at the end of the annealing process nearly correspond to the local minima of the original objective function. At each temperature, a local search heuristics in the continuous domain, which is initialized with the result from the previous temperature step, approaches the closest local minimum. The intuition is to track down a good local minimum, which is hopefully not too far away from the global minimum. In contrast to simulated annealing approaches it is still an open question, whether deterministic annealing approaches can be set up to converge globally.

Deterministic annealing algorithms usually differ regarding the local optimization method. In the following chapters we will discuss a recently developed deterministic annealing approach, which has been independently proposed by Rangarajan et al. [e.g. 24], and Ishii and Sato [11, 12]. The local search algorithm uses the so called soft-assign technique, which we will introduce as well.

CHAPTER 2
A Soft-Assign Quadratic Assignment Algorithm

2.1 Derivation of the Algorithm

2.1.1 History and introduction

This section provides a step-by-step derivation of the *soft-assign quadratic assignment algorithm*[1] for approximately solving quadratic assignment problems, which we will analyze both theoretically and experimentally in the following parts of this work. The early roots of this algorithm can be found in a 1994 paper by Gold et al. [6], as pointed out by [10]. Rangarajan et al. [24] provided the first step-by-step derivation of the algorithm; [7] mentioned the term *soft-assign* for the central technique for the first time. A very similar algorithm was independently proposed by Ishii and Sato in 1996, published as a technical report. A revised version of this report was published in 2001 [12]. Ishii and Sato termed their algorithm *Doubly Constrained Network (DCN)* according to its way

[1]To prevent possible naming confusions, it should be noted here that the terms graduated assignment algorithm, soft-assign QAP algorithm, and soft-assign quadratic assignment algorithm are used as synonyms throughout this work.

of constraint satisfaction. In [10], they noted the similarity between their method and the one from Rangarajan et al. and chose to adopt the term *soft-assign*, because the corresponding techniques were proposed earlier. We will mainly use the term soft-assign for the same reasons.

The soft-assign QAP algorithm belongs to the class of *deterministic annealing algorithms*, which we briefly introduced in section 1.4.2, and therein to the category of continuous state, discrete-time schemes. In the first part of the following derivation we will introduce a deterministic annealing framework for the QAP objective function. The subsequent parts explain the local search that is performed at each computational temperature. Although the particular search algorithm is usually motivated by *artificial neural network approaches*, we omit the relation to neural network algorithms for now and clarify it in chapter 3.

Rangarajan et al. originally derived the algorithm using an unweighted graph matching objective function. We will instead use the more general quadratic assignment objective function in the generalized form by Lawler (cf. equation (1.13)).

Before we begin the derivation, let us restate the optimization problem here. We formulate the problem as minimization of the objective function $E(\boldsymbol{S})$ on the domain of binary matrices. This function directly emerges out of the quadratic assignment problem to be solved[2]:

$$\min_{S\in\{0,1\}^{n\times n}} \quad \underbrace{\frac{1}{2}\sum_{i,j,k,l} C_{ij;kl}S_{ik}S_{jl} + \sum_{i,j} B_{ij}S_{ij}}_{=:E(\boldsymbol{S})} \tag{2.1a}$$

$$\text{subject to} \quad \sum_i S_{ij} = 1, \qquad \forall j \tag{2.1b}$$

$$\sum_j S_{ij} = 1, \qquad \forall i \tag{2.1c}$$

This formulation basically differs from Lawler's form by the explicit row and column sum constraints and a factor of $\frac{1}{2}$ for the quadratic term. This factor simplifies some of the following equations and can be ignored, as long as there is no linear cost. For the time being, we assume $\boldsymbol{C}$ to be symmetric, i. e., $\forall i, j, k, l: \; C_{ij;kl} = C_{ji;lk}$.

2.1.2 Deterministic annealing and barrier function

In this step we integrate the discrete QAP objective function (2.1a) in a continuous state *deterministic annealing framework*. For that, we consider the objective function not only on the domain of binary matrices, but on the domain of positive matrices. As mentioned earlier, deterministic annealing techniques for approximating QAPs are usually based on adding a convex function, which is controlled by the annealing parameter, to the objective function in the continuous domain. Because the QAP objective function itself is usually non-convex, such methods are sometimes referred to as graduated non-convexity methods. The convex function is also called *barrier function*, because of the similarity to barrier function methods in nonlinear programming (see below).

[2]If not denoted otherwise, summations and universal quantifier in this chapter imply the range $\{1, \ldots, n\}$.

When the combinatorial constraint of (2.1) is relaxed into a positivity constraint, we obtain the following continuous problem:

$$\begin{aligned}
\min_{\boldsymbol{M}\in\mathbb{R}^{n\times n}} \quad & \frac{1}{2}\sum_{i,j,k,l} C_{ij;kl} M_{ik} M_{jl} + \sum_{i,j} B_{ij} M_{ij} && \text{(2.2a)}\\
\text{subject to} \quad & \sum_i M_{ij} = 1, \qquad \forall j && \text{(2.2b)}\\
& \sum_j M_{ij} = 1, \qquad \forall i && \text{(2.2c)}\\
& M_{ij} \geq 0, \qquad \forall i,j && \text{(2.2d)}
\end{aligned}$$

Such positive matrices, for which each row and each column sums to one, are called *doubly stochastic*.

A nonlinear programming problem of minimizing $\Phi(\boldsymbol{x})$ under the constraints $\boldsymbol{Ax} = \boldsymbol{b}$ and $g_i(\boldsymbol{x}) \geq 0$ can be handled using a barrier function approach (see, e. g., Murthy [20, pp. 469f]). Such a technique enforces the inequality constraints $g_i(\boldsymbol{x}) \geq 0$, which specify a subdomain, using a barrier function in an iterative algorithm.

The r-th iteration step of the barrier function method consists of solving the modified minimization problem $\Phi(\boldsymbol{x}) - \alpha^{(r)} \sum_i \log g_i(\boldsymbol{x})$ under the constraint $\boldsymbol{Ax} = \boldsymbol{b}$; let us call the result $\boldsymbol{x}^{(r)}$. The algorithm for solving the modified problem is commonly based on some directional descent, which starts at $\boldsymbol{x}^{(r-1)}$, the result of the previous iteration. Variables that approach 0 during the minimization are pushed back toward higher values by the strongly increasing gradient of the barrier function (The gradient tends to ∞, when x approaches 0). We should note here that the modified problem is, as we intended, no longer explicitly restricted by the inequality constraints. When the iterative process is initialized with a feasible solution $\boldsymbol{x}^{(0)}$ and carried out with a monotonically decreasing barrier parameter $\alpha^{(r)}$ it converges, under mild assumptions, to the solution of the original problem.

Let us apply this technique formally to the above objective function (2.2a), albeit it is not used to solve the problem exactly: The sum constraints are assumed to be expressed as a linear equation and the function g is set to identity. For the deterministic annealing framework, we replace the barrier function $\log x$ by a negative entropy function $-H(x) = -x \log x$. The negative entropy function, as opposed to the traditional logarithmic barrier function is used here, because it will allow us to show certain properties regarding the convergence in section 2.5.2. Rangarajan et al. [26] reported that similar results would be hard, if not impossible, to obtain for other barrier functions including the logarithmic one. As for the logarithmic function, the gradient of the negative entropy function tends toward infinity when its argument approaches 0. Strictly speaking, the negative entropy function is not a barrier function, because it does not tend to infinity when the argument approaches 0 [cf. 18]. Nevertheless, it can be used in the same way as the pure logarithmic barrier function. Furthermore, we observe that the entropy function is convex on $\mathbb{R}_+$ by considering

$$\frac{d}{dx} H(x) = \log x + 1 \tag{2.3}$$

and

$$\frac{d^2}{dx^2}H(x) = \frac{1}{x} > 0, \qquad \forall x > 0. \tag{2.4}$$

Thus, the entropy function $H(x) = x \log x$, weighted with the annealing parameter $\frac{1}{\beta}$, which corresponds to the barrier parameter, can be used to build a deterministic annealing framework. To control all the state variables, we add the term $\frac{1}{\beta}\sum_{ij} M_{ij}\log(M_{ij})$ to the continuous objective function (2.2a). After removing the now redundant positivity constraint (2.2d), we obtain the desired deterministic annealing framework, in which an annealing parameter β is used to control the convexity of the compound objective function. Low values make the function nearly convex and therefore easier to minimize, whereas high values of β make the problem more similar to the original (QAP) problem. Furthermore, the barrier function ensures the positivity of every continuous state variable M_{ij}. Because of the sum constraint satisfaction, this term also puts an upper bound on the M_{ij}, since a value greater than 1 would require another value to be negative, what is not allowed.

2.1.3 Self-amplification

Experimentally, several publications [e. g., 12, 22, 26] reported two-cycle oscillations for certain instances, which can be removed with a *self-amplification term*. This term is added to the continuous objective function. It influences the *continuous* objective function such that the global minimum (or the global minima) is identical to the one of the *combinatorial* QAP objective function[3]. We will furthermore need this term to show some convergence related properties of the local search algorithm (cf. section 2.5).

The self-amplification term $-\frac{\gamma}{2}\sum_{i,j} M_{ij}^2$, which we use here, has the property that is does not modify the minimum the combinatorial objective function, because the term is constant for permutation matrices and could thus be omitted. Nevertheless, when doubly stochastic matrices are considered, the term is no longer a constant and can therefore modify the dynamics of the algorithm. To simplify the remaining part of the derivation, we introduce the self-amplification term by defining a new compound cost matrix or array

$$\boldsymbol{C}^{(\gamma)} = \left(C^{(\gamma)}_{ij;kl}\right)^{n}_{i,j,k,l=1}, \qquad C^{(\gamma)}_{ij;kl} := \begin{cases} C_{ij;kl}, & i \neq j \ \vee \ k \neq l \\ C_{ii;kk} - \gamma, & \text{otherwise.} \end{cases} \tag{2.5}$$

For now, it is sufficient for us to know that γ is set up such that $\boldsymbol{C}^{(\gamma)}$ is negative definite, which makes the quadratic term concave. Therefore, the compound objective is no relaxation of the QAP objective for which computing the optimum is tractable. In section 2.3.1 we will explain this setup and its relation to the above mentioned goals.

[3]This only holds for the limit of β approaching ∞.

2.1.4 Algebraic transformation

To carry out the deterministic annealing algorithm, we have to approach the closest local minimum of the continuous objective function at every temperature on the annealing schedule. This is done using a local search algorithm based on a discrete-time update scheme, which we will derive in this and the following sections.

Using the two aforementioned techniques, we can formulate a new, continuous problem, which is to be minimized at each temperature on the still to be defined annealing schedule[4]:

$$\min_{\boldsymbol{M}\in(0:1)^{n\times n}} \underbrace{\frac{1}{2}\sum_{i,j,k,l} C^{(\gamma)}_{ij;kl} M_{ik} M_{jl} + \sum_{i,j} B_{ij} M_{ij} + \frac{1}{\beta}\sum_{i,j} M_{ij}\log(M_{ij})}_{=:F(\boldsymbol{M})} \tag{2.6a}$$

$$\text{subject to} \quad \sum_i M_{ij} = 1, \qquad \forall j \tag{2.6b}$$

$$\sum_j M_{ij} = 1, \qquad \forall i. \tag{2.6c}$$

Our first step in handling the preceding minimization problem is a transformation of the objective function $F(\boldsymbol{M})$, in which the quadratic term is split up. According to Mjolsness and Garrett [19], the minimization of an objective function $f(x) = \frac{1}{2}x^2$ under some constraints is equivalent to the simultaneous minimization of the objective function $\hat{f}(x,\sigma) = x\sigma - \frac{1}{2}\sigma^2$ under the same constraints. This results out of the fact that the fixed points regarding x are preserved. After applying the *algebraic transformation* to each relevant summand of the above objective function, we obtain

$$F(\boldsymbol{M},\boldsymbol{X}) = \sum_{i,j,k,l} C^{(\gamma)}_{ij;kl} M_{ik} X_{jl} - \frac{1}{2}\sum_{i,j,k,l} C^{(\gamma)}_{ij;kl} X_{ik} X_{jl} + \sum_{i,j} B_{ij} M_{ij} + \frac{1}{\beta}\sum_{i,j} M_{ij}\log(M_{ij}). \tag{2.7}$$

The resulting function is clearly linear in $\boldsymbol{M}$ (when disregarding the barrier function), but quadratic in $\boldsymbol{X}$. For the graduated assignment algorithm we use these properties to built an iteration scheme, which performs alternate minimizations instead of simultaneous minimizations. For now, we disregard the sum constraints and just consider the stationary conditions of equation (2.7):

$$\frac{\partial}{\partial X_{jl}} F(\boldsymbol{M},\boldsymbol{X}) = \sum_{i,k} C^{(\gamma)}_{ij;kl} M_{ik} - \sum_{i,k} C^{(\gamma)}_{ij;kl} X_{ik} = 0, \qquad \forall j,l \tag{2.8a}$$

$$\frac{\partial}{\partial M_{ik}} F(\boldsymbol{M},\boldsymbol{X}) = \sum_{j,l} C^{(\gamma)}_{ij;kl} X_{jl} + B_{ik} + \frac{1}{\beta}\left(\log(M_{ik}) + 1\right) = 0, \qquad \forall i,k \tag{2.8b}$$

Since $\boldsymbol{C}^{(\gamma)}$ is negative definite, minimizing $F(\boldsymbol{M},\boldsymbol{X})$ with respect to $\boldsymbol{X}$, while keeping the other arguments fixed, *uniquely* yields $\boldsymbol{X} = \boldsymbol{M}$. Minimizing with respect to $\boldsymbol{M}$, while the keeping the other variables fixed, yields

$$M_{ik} = \exp\left[-\beta\left(\sum_{j,l} C^{(\gamma)}_{ij;kl} X_{jl} + B_{ik}\right) - 1\right], \qquad \forall i,k. \tag{2.9}$$

[4]We may restrict the search space to the interval $(0:1)^{n\times n}$, because it is assumed that the local search starts at a feasible point inside this domain, which will furthermore not be left during the minimization.

The idea, which later yields the discrete-time update scheme, is that these separate minimizations can be performed in an alternating way. The intuition is that each of the separate minimization steps lowers the objective function value. Above, however, we ignored the satisfaction of the row and column sum constraints. Thus, a technique for constraint satisfaction has to be added before we set up the iterative update scheme.

2.1.5 Hard constraint satisfaction

In contrary to previous deterministic annealing approaches, e. g., from Peterson and Söderberg [22], the soft-assign QAP algorithm strives to encode the row and column sum constraints directly into the minimization algorithm, as opposed to modifying the objective function. Previous deterministic annealing approaches used "soft" constraint satisfaction methods that use a penalty term in the objective function. Such a penalty term allows undesirable configurations that do not satisfy the constraints, but penalizes them. In other words, soft constraint satisfaction induces a configuration space that is larger than the solution space and thus contains redundant configurations, whereas for *hard constraint satisfaction* the configuration space is equal to the solution space. The latter approach is strongly favorable, since the redundancy of soft constraint satisfaction approaches has been shown to induce adverse scaling properties; i. e., the solution quality generally becomes worse, when the size of the problem increases. [22] reported that the negative effect increases for hard, random problems and is less severe for easy, structured problems.

To integrate the constraint satisfaction in the minimization process, let us introduce *Lagrange* multipliers to enforce the row and column sum constraints. Later we shall see that these constraints can be satisfied in an elegant way. We obtain the corresponding Lagrange function

$$L(\boldsymbol{M},\boldsymbol{X},\boldsymbol{\mu},\boldsymbol{\nu}) = F(\boldsymbol{M},\boldsymbol{X}) + \sum_i \mu_i \left(\sum_j M_{ij} - 1 \right) + \sum_j \nu_j \left(\sum_i M_{ij} - 1 \right), \tag{2.10}$$

where the μ_i and the ν_j are the Lagrange multipliers. The introduction of Lagrange multipliers for this specific problem was already proposed by Yuille and Kosowsky [32] in the context of continuous time frameworks. They suggested a gradient descent on the objective function combined with a projection of the gradient onto the space obeying the constraints. Rangarajan et al. [24] reported this procedure to be problematic regarding implementations on digital computers and to be algorithmically inefficient. Here, a different path is followed, for which we first consider the stationary conditions of $L(\boldsymbol{M}, \boldsymbol{X}, \boldsymbol{\mu}, \boldsymbol{\nu})$:

$$\frac{\partial}{\partial X_{jl}} L(\boldsymbol{M}, \boldsymbol{X}, \boldsymbol{\mu}, \boldsymbol{\nu}) = \sum_{i,k} C^{(\gamma)}_{ij;kl} M_{ik} - \sum_{i,k} C^{(\gamma)}_{ij;kl} X_{ik} = 0, \quad \forall j, l \tag{2.11a}$$

$$\frac{\partial}{\partial M_{ik}} L(\boldsymbol{M}, \boldsymbol{X}, \boldsymbol{\mu}, \boldsymbol{\nu}) = \sum_{j,l} C^{(\gamma)}_{ij;kl} X_{jl} + B_{ik} + \mu_i + \nu_k + \frac{1}{\beta} (\log(M_{ik}) + 1) = 0, \quad \forall i, k \tag{2.11b}$$

$$\frac{\partial}{\partial \mu_i} L(\boldsymbol{M}, \boldsymbol{X}, \boldsymbol{\mu}, \boldsymbol{\nu}) = \sum_{j} M_{ij} - 1 = 0, \quad \forall i \tag{2.11c}$$

$$\frac{\partial}{\partial \nu_j} L(\boldsymbol{M}, \boldsymbol{X}, \boldsymbol{\mu}, \boldsymbol{\nu}) = \sum_{i} M_{ij} - 1 = 0, \quad \forall j. \tag{2.11d}$$

Equation (2.11b) is then reformulated as

$$M_{ik} = \exp\left[-\beta\left(\sum_{j,l} C^{(\gamma)}_{ij;kl} X_{jl} + B_{ik} + \mu_i + \nu_k\right) - 1\right], \qquad \forall i, k, \tag{2.12}$$

which will be the basis for the discrete-time algorithm. We again rewrite the equation as

$$M_{ik} = \underbrace{\exp(-\beta\mu_i)}_{=:u_i}\underbrace{\exp(-\beta\nu_k)}_{=:v_k}\underbrace{\exp\left[-\beta\left(\sum_{j,l} C^{(\gamma)}_{ij;kl} X_{jl} + B_{ik}\right) - 1\right]}_{=:W_{ik}}, \qquad \forall i, k. \tag{2.13}$$

We observe from this equation that the u_i essentially scale the i-th row of $\boldsymbol{W}$ and v_k the k-th column of this matrix. If the stationary conditions (2.11c) and (2.11d) are required to hold as well, the Lagrange parameters hence scale the strictly positive matrix $\boldsymbol{W}$ (due to the exponential function) to the doubly stochastic matrix $\boldsymbol{M}$. We can now rewrite equation (2.13) in vectorial form giving

$$\boldsymbol{M} = \operatorname{diag}\{\boldsymbol{u}\} \boldsymbol{W} \operatorname{diag}\{\boldsymbol{v}\}. \tag{2.14}$$

As pointed out, e. g., in [7, 24], a theorem by Sinkhorn [28] provides a convenient way of obtaining these scale factors:

Theorem 2.1 (Sinkhorn)
To a given strictly positive $n \times n$ matrix $\boldsymbol{A}$ there corresponds exactly one doubly stochastic matrix $\boldsymbol{T}_A$, which can be expressed in the form $\boldsymbol{T}_A = \boldsymbol{D}_1 \boldsymbol{A} \boldsymbol{D}_2$, where $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ are diagonal matrices with positive diagonals. The matrices $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ are themselves unique up to a scalar factor.

Proof. See [28].

Theorem 2.2 (Sinkhorn)
The iterative process of alternately normalizing the rows and columns of a strictly positive $n \times n$ matrix is convergent to a strictly positive doubly stochastic matrix.

Proof. See [28].

From the vectorial form in equation (2.14), it is easy for us to observe that the so called Sinkhorn balancing from theorem 2.2 is able to provide the unique doubly stochastic matrix $\boldsymbol{M}$. Thus the Sinkhorn balancing procedure solves for the Lagrange parameter vectors $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ without explicitly providing them. Considering that alternate normalizations can be easily carried out, they are well suited for minimization of the objective under the sum constraints. If other barrier functions are to be used in conjunction with the Sinkhorn balancing, it has to be verified, whether the positivity requirement of the balancing procedure still holds for the particular barrier function.

The *soft-assign technique* subsumes the Sinkhorn scaling operations in conjunction with the computational temperature and the exponential function. We assume $\boldsymbol{Z}$ to be a cost matrix induced by some still to be specified problem:

$$M_{ik}^{(0)} := \exp(\beta Z_{ik}), \qquad \forall i,k \tag{2.15a}$$

$$M_{ik}^{(2s-1)} := \frac{M_{ik}^{(2s-2)}}{\sum_j M_{ij}^{(2s-2)}}, \qquad \forall i,k; s \geq 1 \tag{2.15b}$$

$$M_{ik}^{(2s)} := \frac{M_{ik}^{(2s-1)}}{\sum_j M_{jk}^{(2s-1)}}, \qquad \forall i,k; s \geq 1 \tag{2.15c}$$

The preceding technique enforces two types of sum constraints, namely that each row and each column sums to one. If the above cost matrix $\boldsymbol{Z}$ is the cost matrix of a linear assignment problem, the soft-assign technique can be used to solve linear assignment problems when it is combined with deterministic annealing. This was first reported by Kosowsky and Yuille [15]. Because of this property, Gold and Rangarajan [8] termed the method *soft-assign*. A similar technique, which only enforces one sum constraint, is the *soft-max* [see 8], which can, for example, be used to find the maximum of a discrete set.

2.1.6 Discrete-time synchronous update scheme

After the method of constraint satisfaction was introduced, we finally assemble an iteration scheme and with that our local search heuristics. Starting from the algebraic transformation and adding row and column sum constraint satisfaction, we derived a set of simultaneous equations (2.11a) - (2.11d), which will be the basis for the *discrete-time synchronous update scheme* below.

As already mentioned in section 2.1.4, we can define a discrete-time update scheme by minimizing the objective regarding the different variables separately in an alternating way. For that, a superscript is first added to each argument of the Lagrange function, which indicates the current discrete time step. Second, we minimize $L\left(\boldsymbol{M}^{(r-1)}, \boldsymbol{X}^{(r-1)}\right)$ with respect to $\boldsymbol{X}^{(r-1)}$, while keeping the other arguments fixed. According to the results of section 2.1.4, the stationary condition (2.11a) leads us to the update step

$$\boldsymbol{X}^{(r)} := \boldsymbol{M}^{(r-1)}, \qquad \forall r \geq 1. \tag{2.16}$$

The next update step minimizes $L\left(\boldsymbol{M}^{(r-1)}, \boldsymbol{X}^{(r)}\right)$ with respect to $\boldsymbol{M}^{(r-1)}$ under exact sum constraint satisfaction, while keeping the other variables fixed. As we have shown in the previous section, the remaining stationary conditions (2.11b) - (2.11d) can be fulfilled simultaneously by using the soft-assign method. With equation (2.12) we define the second update step as

$$M_{ik}^{(r)} := \exp\left[-\beta\left(\sum_{j,l} C_{ij;kl}^{(\gamma)} X_{jl}^{(r)} + B_{ik} + \mu_i^{(r)} + \nu_k^{(r)}\right) - 1\right], \qquad \forall i,k; r \geq 1, \tag{2.17}$$

where it is assumed that $\boldsymbol{\mu}^{(r)}$ and $\boldsymbol{\nu}^{(r)}$ are set up using Sinkhorn's algorithm such that $\boldsymbol{M}^{(r)}$ has the property of rows and columns summing to one. This iteration scheme falls into the class of synchronous update schemes, since the state variables are updated synchronously using only values from the previous iteration step, as opposed to serial update schemes, where only one state variable is updated at a time using partly previous and partly current state variables.

Before finally describing the algorithm, we should note another important interpretation of the preceding derivation. Gold and Rangarajan [7] alternatively described the scheme by considering a Taylor series expansion of the continuous, but otherwise unmodified objective function at time step r using the $(r-1)$-th time step as initial condition. By omitting the linear term, we receive the following first order approximation:

$$\begin{aligned}\frac{1}{2}\sum_{i,j,k,l} C_{ij;kl}^{(\gamma)} M_{ik}^{(r)} M_{jl}^{(r)} \approx & \frac{1}{2}\sum_{i,j,k,l} C_{ij;kl}^{(\gamma)} M_{ik}^{(r-1)} M_{jl}^{(r-1)} \\ & + \sum_{i,k}\left[\left(\sum_{j,l} C_{ij;kl}^{(\gamma)} M_{jl}^{(r-1)}\right)\cdot\left(M_{ik}^{(r)} - M_{ik}^{(r-1)}\right)\right]\end{aligned} \tag{2.18}$$

Minimizing the right-hand side of (2.18) regarding $\boldsymbol{M}^{(r)}$ is equivalent to solving a linear assignment problem (we again add the previously omitted linear cost):

$$\min_{\boldsymbol{M}^{(r)}\in\{0,1\}^{n\times n}} \sum_{i,k} \underbrace{\left[\sum_{j,l} C_{ij;kl}^{(\gamma)} M_{jl}^{(r-1)} + B_{ik}\right]}_{=:-Z_{ik}^{(r)}} M_{ik}^{(r)}, \qquad \forall r \geq 1 \tag{2.19a}$$

$$\text{subject to} \quad \sum_i M_{ij}^{(r)} = 1, \qquad \forall j; r \geq 1 \tag{2.19b}$$

$$\sum_j M_{ij}^{(r)} = 1, \qquad \forall i; r \geq 1 \tag{2.19c}$$

In summary, the Taylor series approximation yields a local, *linear relaxation* of the quadratic assignment problem, which can be solved using the soft-assign technique.

2.2 The Algorithm

2.2.1 Introduction

Before the whole algorithm is described, we have to add a few, but nevertheless important details.

When we derived the deterministic annealing framework, we already mentioned that the descent has to start at an admissible configuration (i. e., a doubly stochastic matrix) in the interior of the admissible subdomain. We should first note that any scaling factor that is applied to every element of the initial assignment matrix is removed after the first row normalization. Thus, the absolute scaling is irrelevant. As we shall see in section 2.3.2, the objective has a single trivial fixed point at $\frac{1}{n}$ for sufficiently high temperatures, to which the update scheme probably converges. Hence, initially setting every element to $\frac{1}{n}$ immediately brings the system into its equilibrium. If the initial temperature, in contrary, is not high enough, it is favorable to add symmetric noise to the values (for example from a Gaussian distribution), to prevent the system from converging to the trivial local minimum. Because adding noise in the former case does not prevent convergence, there is no drawback in always doing so.

Furthermore, we have to define an annealing schedule for the inverse temperature parameter β. We assume an initial inverse temperature β_0 to be given, which will be determined later. At each temperature step, the minimization of the modified objective function is performed with the discrete-time update scheme described in the previous section. After approximate convergence (if the scheme converges), we rise the inverse temperature through multiplication by a constant factor β_r, the annealing rate. The annealing process ends at a given inverse temperature β_f.

Following the above observation that any common scaling factor of the assignment matrix is removed after the first row normalization, the Lagrange parameter vectors may initially be set to zero and the subtraction of 1 in update step (2.17) may be omitted as well.

For the practical implementation, we have to establish convergence criteria for the discrete-time update scheme. This is achieved by stopping the scheme, when the change of the assignment matrix is sufficiently low. Here, we test the change, measured by the Frobenius matrix norm, against a given threshold δ. Moreover, we also impose a strict bound on the number of iterations (r_f), which prevents infinite loops in the case of divergence (e. g., because of adverse parameter settings).

Finally, let us define convergence bounds for the Sinkhorn balancing procedure. The alternate normalizations are performed until the maximum deviation of the row sums from 1 does not exceed a given threshold ϵ. Again, we limit the number of iterations by a constant (s_f).

2.2.2 The soft-assign quadratic assignment algorithm

Combining all previous techniques and components leads us to the soft-assign quadratic assignment algorithm 2.1. Table 2.1, which is given below, summarizes the symbols used for describing the algorithm.

Algorithm 2.1 Soft-Assign Quadratic Assignment Algorithm

```
    $\beta \leftarrow \beta_0$
    $M_{ik}^{(0)} \leftarrow \frac{1}{n} \cdot \mathcal{N}(1, 0.1), \quad \forall i, k$
    while $\beta \leq \beta_f$ do {Deterministic annealing}
      $r \leftarrow 0$
 5:   repeat
        $r \leftarrow r + 1$
        $Z_{ik}^{(r)} \leftarrow -\left[\sum_{j,l} C_{ij;kl}^{(\gamma)} M_{jl}^{(r-1)} + B_{ik}\right], \quad \forall i, k$    {Linear relaxation}
        $M_{ik}^{(r,0)} \leftarrow \exp\left(\beta Z_{ik}^{(r)}\right), \quad \forall i, k$    {Soft-assign}
        $s \leftarrow 0$
10:     repeat {Sinkhorn balancing}
          $s \leftarrow s + 1$
          $M_{ik}^{(r,2s-1)} \leftarrow \frac{M_{ik}^{(r,2s-2)}}{\sum_j M_{ij}^{(r,2s-2)}}, \quad \forall i, k$    {Row normalization}
          $M_{ik}^{(r,2s)} \leftarrow \frac{M_{ik}^{(r,2s-1)}}{\sum_j M_{jk}^{(r,2s-1)}}, \quad \forall i, k$    {Column normalization}
        until $\max_i |\sum_k M_{ik}^{(r,2s)} - 1| < \epsilon \quad \vee \quad s > s_f$
15:     $M_{ik}^{(r)} \leftarrow M_{ik}^{(r,2s)}, \quad \forall i, k$
      until $\frac{\|\boldsymbol{M}^{(r)} - \boldsymbol{M}^{(r-1)}\|_F}{N} < \delta \quad \vee \quad r > r_f$
      $M_{ik}^{(0)} \leftarrow M_{ik}^{(r)}, \quad \forall i, k$
      $\beta \leftarrow \beta\beta_r$
    end while
20: Clean-up heuristics
```

A running time analysis of algorithm 2.1 reveals that it is polynomially bounded: The number of iterations of the loops beginning in lines 3, 5, and 10 are all bounded by constants. Thus only the matrix calculations contribute to the order of computational complexity. They all have $\mathcal{O}(n^2)$ worst-case complexity, except line 7, which has $\mathcal{O}(n^4)$ worst-case complexity. We can further improve this bound by exploiting the sparse structure of the quadratic cost matrix $\boldsymbol{C}$. A careful implementation will be able to bound the running time by $\mathcal{O}(\max\{n, |V_1|\} \cdot \max\{n, |V_2|\})$.

Symbol	*Meaning*
$\boldsymbol{C}^{(\gamma)}$	quadratic cost matrix including self-amplification term
$\boldsymbol{B}$	linear cost matrix
$\boldsymbol{Z}^{(r)}$	effective cost matrix at relaxation step r
$\boldsymbol{M}^{(r,2s)}$	assignment matrix at relaxation step r and Sinkhorn balancing step s
β	current inverse temperature
β_0	initial inverse temperature
β_r	annealing rate
β_f	final inverse temperature
r_f	maximum number of relaxation steps at each temperature step
s_f	maximum number of Sinkhorn balancing steps at each relaxation step
δ	approximate convergence threshold for relaxation
ϵ	approximate convergence threshold for Sinkhorn balancing

Table 2.1: Variables and constants of the soft-assign quadratic assignment algorithm

2.2.3 Clean-up heuristics

In anticipation of section 2.5, let us note here that the annealing algorithm, under certain assumptions, probably converges arbitrarily close to a permutation matrix. But a rough annealing schedule, the hard iteration bounds, and numerical inaccuracies may prevent the algorithm from returning a permutation matrix. Instead, a nearly doubly stochastic matrix is returned, which has to be transformed into a permutation matrix. Gold and Rangarajan [7] proposed to binarize the matrix entries by setting the largest value in each row to 1, the rest to 0. This method yields a permutation matrix, if the maxima occur in different columns, or in other words if the doubly stochastic matrix is row dominant. But there are certain situations, where this is not the case. Experimentally, symmetries in the cost matrix were observed to cause the obtained doubly stochastic matrix to have multiple largest entries in each row, which moreover appear in several rows. We can, for example, circumvent this problem by iteratively choosing one of the multiple largest entries in a not yet occupied column. Alternatively, a linear assignment problem can be put up, which finds the closest permutation matrix [7].

Ishii and Sato [12] proposed to use a *2opt heuristics* to perform a local optimization of the obtained permutation matrix. This greedy algorithm is especially known for approximately solving traveling salesman problems [4, pp. 246f] and is also mentioned in the context of quadratic assignment problems by Burkard et al. [2, pp. 9f]. The 2opt method belongs to the class of improvement methods, which perform a search on the local neighborhood of feasible solutions. For this particular method, the pair-exchange neighborhood is considered, i. e., the configurations that are reached by exchanging pairs of assignments. The permutation is traversed (possibly multiple times) in a predefined order, whereby the first encountered pair, whose exchange improves the QAP objective function, is actually exchanged. This is done until no further improvement is possible. We should note here, that this algorithm has been shown to have exponential worst-case running time [2, p. 10], but performs quite well in nearly all practical cases. Sections 4.2.1, 4.2.2, and 4.3 give experimental results on the improvement obtained by applying the 2opt greedy strategy.

2.3 Theoretically Motivated Parameter Choice

The dynamics of the soft-assign quadratic assignment algorithm is influenced by several algorithm parameters, but only two of them were successfully studied theoretically. At first, this section discusses the influence of the self-amplification parameter γ as introduced in section 2.1.3. Later, the influence of the initial inverse temperature on the dynamics is studied.

2.3.1 Self-amplification parameter

We established the self-amplification term, controlled by a parameter γ, to improve the solution quality. Let us now study the effectiveness of this term. Yuille and Kosowsky [32] introduced it to ensure that the global minimum of the continuous objective function corresponds to the global minimum of the combinatorial QAP optimization problem[5]. They derived a general criterion for the self-amplification parameter, for which they use the following figurative imagination: The admissible domain of doubly stochastic matrices can be seen as part of the interior of a n^2-dimensional unit hypercube $\{0,1\}^{n^2}$. The vertices of this hypercube correspond to the domain of binary matrices; parts of these vertices form the domain of permutation matrices. Yuille and Kosowsky in other words wanted to guarantee that the minimum of the continuous objective function lies at one of the admissible vertices of this hypercube, as it does for the combinatorial objective. This is achieved by shifting the eigenvalues of $\boldsymbol{C}$ toward the negative, such that the interior of the hypercube has at least one negative eigenvalue. Moreover, minima on the faces of the hypercube also have to be prevented. They finally give the following theorem:

Theorem 2.3 (Yuille and Kosowsky)
Let the objective function be of the form $E(\boldsymbol{M}) = \sum_{i,j,k,l} C_{ij;kl} M_{ik} M_{jl}$. Then one can ensure that the minima lie at the vertices of the hypercube by adding a term $-\gamma \sum_{i,k} M_{ik}^2$ where $\gamma > \|\boldsymbol{C}\|_{\mathrm{F}}$.

Proof. See Yuille and Kosowsky [32].

Setting up γ according to this criterion implies that $\boldsymbol{C}^{(\gamma)}$ is negative definite, since $\|\boldsymbol{C}\|_{\mathrm{F}}$ is an upper bound on the absolute value of the eigenvalues of $\boldsymbol{C}$. Although this guarantees that the global minimum of the continuous objective function is identical to the one of the discrete objective function, we shall see that the convergence related properties in section 2.5 have less restrictive requirements. For these convergence studies, the negative definiteness of $\boldsymbol{C}^{(\gamma)}$ may be restricted to the subspace of matrices with columns summing to zero. In anticipation of this, let us derive a criterion that is suitable

[5]If the global minimum is not unique, the statements still apply analogously.

in practice. We will use the following lemma informally proposed by Rangarajan et al. [25]:

Lemma 2.4
Let the matrix $\boldsymbol{T}_n$ be defined as $\boldsymbol{T}_n := \boldsymbol{I}_n - \frac{1}{n}\mathbf{1}_{n\times n}$. For an arbitrary, real matrix $\boldsymbol{M}$ the transformed matrix $\hat{\boldsymbol{M}} := \boldsymbol{T}_n\boldsymbol{M}$ has annihilated column sums.

Proof. See section A.3.

We should recall that $\boldsymbol{C}^{(\gamma)}$ is strictly negative definite on the subspace of matrices with annihilated column sums, if and only if

$$\operatorname{vec}(\boldsymbol{M})^{\mathrm{T}}\boldsymbol{C}^{(\gamma)}\operatorname{vec}(\boldsymbol{M}) < 0, \qquad \forall \boldsymbol{M} \in \mathbb{R}^{n\times n}, \quad \forall j : \sum_i M_{ij} = 0. \tag{2.20}$$

Using the property from lemma 2.4 and subsequently the Kronecker product properties from section A.2, we can rewrite this as

$$\begin{gathered}\operatorname{vec}(\boldsymbol{T}_n\boldsymbol{M}\boldsymbol{I}_n)^{\mathrm{T}}\boldsymbol{C}^{(\gamma)}\operatorname{vec}(\boldsymbol{T}_n\boldsymbol{M}\boldsymbol{I}_n) = \\ \operatorname{vec}(\boldsymbol{M})^{\mathrm{T}}(\boldsymbol{I}_n\otimes\boldsymbol{T}_n)\boldsymbol{C}^{(\gamma)}(\boldsymbol{I}_n\otimes\boldsymbol{T}_n)\operatorname{vec}(\boldsymbol{M}) < 0, \qquad \forall \boldsymbol{M}\in\mathbb{R}^{n\times n}.\end{gathered} \tag{2.21}$$

What follows is a convenient criterion for negative definiteness on the above subspace: $\boldsymbol{C}^{(\gamma)}$ is strictly negative definite on the subspace of matrices with columns summing to zero, if and only if

$$\lambda_{\max}\Big((\boldsymbol{I}_n\otimes\boldsymbol{T}_n)\boldsymbol{C}^{(\gamma)}(\boldsymbol{I}_n\otimes\boldsymbol{T}_n)\Big) < 0. \tag{2.22}$$

If γ is chosen such that the inequality (2.22) holds, this allows us to show certain properties regarding the convergence, which imply that the self-amplification term removes the stability problems. In practice, however, the self-amplification parameter may be chosen empirically, as it is done for most of the experiments documented in chapter 4, because we shall see that a proper choice is crucial to optimal performance (cf. section 4.2.3).

2.3.2 Critical inverse temperature

Second, we examine the initial inverse temperature of the annealing schedule and its influence on the dynamics of the algorithm. When the computational temperature parameter $T = \frac{1}{\beta}$ is high, the convex barrier function causes the modified objective function to be nearly convex. If we only minimize the barrier function under row and column sum constraints, we obtain that the trivial fixed point $M_{ij} = \frac{1}{n}$, $\forall i, j$ is the unique global minimum (cf. section A.4). If the algorithm converges to a local minimum, it also converges into a unique global minimum. Hence, we would like to start the annealing at a sufficiently high temperature, where the algorithm still converges to a unique global minimum. This guarantees that the algorithm does not "miss" important bifurcations, which emerge at lower temperatures through the multiple local minima of the QAP objective. In the following, we will estimate the inverse critical temperature β_c, at which the first bifurcation occurs. Ishii and Sato [12], Peterson and Söderberg

[22], Yuille and Kosowsky [32] gave similar conclusions. Let us first consider the Hessian of the continuous objective function, whereby the Lagrange terms are disregarded:

$$\frac{\partial^2}{\partial M_{ik} \partial M_{jl}} E(\boldsymbol{M}) = C_{ij;kl} + \left(-\gamma + \frac{1}{\beta M_{ik}} \right) \delta_{ij} \delta_{kl}, \qquad \forall i, j, k, l \tag{2.23}$$

The trivial fixed point $M_{ik} = \frac{1}{n}$ is now used to determine the critical temperature, although the fixed point might be slightly deviating, because of the increasing influence of the actual QAP objective. By requiring the Hessian to be positive definite for this particular $\boldsymbol{M}$, we estimate the critical temperature as

$$T_\mathrm{c} = \frac{1}{\beta_\mathrm{c}} > \max \left\{ 0, \frac{1}{n} \left(-\lambda_{min}(\boldsymbol{C}) + \gamma \right) \right\} = \max \left\{ 0, -\frac{1}{n} \lambda_{min} \left(\boldsymbol{C}^{(\gamma)} \right) \right\}. \tag{2.24}$$

Altogether, this gives us a feasible criterion for ensuring that the initial temperature exceeds the critical temperature.

Bifurcations

Ishii and Sato analyzed the problem related, as well as the algorithm related bifurcation structure. For generic quadratic assignment problems, symmetries and corresponding bifurcations do not generally exist [11]. Problem specific analyses can only be made for certain subclasses of the QAP, e. g., for TSP problems, for which [27] gives an in-depth review. Regarding algorithm related bifurcations, Ishii and Sato [12] reported a significantly improved experimental behavior compared to other deterministic annealing approaches with soft constraint satisfaction.

2.4 Handling Special Cases with the Algorithm

2.4.1 Imbalanced problems

In some cases it is desirable to compute an assignment between two sets with different cardinality, which we refer to as rectangular or imbalanced problems. For that, the objective is to assign the smaller set to an equal-sized subset of the larger set. This requires not only finding the best assignment to a specific subset, but also finding the subset that yields the best overall assignment. Especially graph matching problems posed by view-based object recognition problems may lead to such asymmetric situations, since, for example, occlusion may prevent the detection of some of the features used and hence results in unequally-sized sets.

Let us define the imbalanced quadratic assignment problem using n facilities and m locations and corresponding square cost matrices $\boldsymbol{F} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{D} \in \mathbb{R}^{m \times m}$. The linear costs are given by $\boldsymbol{B} \in \mathbb{R}^{n \times m}$. To simplify further steps, we may suppose that $n \geq m$, otherwise the sets to be matched are exchanged together with their cost matrices. Because of unequally-sized sets, we can no longer assume the assignment to be a permutation matrix. Instead some equality constraints have to be relaxed to inequality constraints. We now define match matrices as

$$\tilde{\Pi}_{n,m} := \left\{ \boldsymbol{X} \in \{0,1\}^{n \times m} \;\middle|\; \forall j : \sum_{i=1}^{n} X_{ij} = 1,\ \forall i : \sum_{j=1}^{m} X_{ij} \leq 1 \right\}. \tag{2.25}$$

Similar to the usual definition of quadratic assignment problems, we define the imbalanced QAP using the trace formulation:

$$\boldsymbol{X}^* = \arg \min_{\boldsymbol{X} \in \tilde{\Pi}_{n,m}} \operatorname{tr} \left\{ \boldsymbol{F}\boldsymbol{X}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}} + \boldsymbol{B}\boldsymbol{X}^{\mathrm{T}} \right\} \tag{2.26}$$

From now on we ignore the linear cost, as it can be handled quite easily.

When the traditional graph matching objective is extended to graphs of different size, it is unfortunately no longer a quadratic assignment problem. We can see this by considering

$$\begin{aligned} \|\boldsymbol{X}\boldsymbol{A}_1\boldsymbol{X}^{\mathrm{T}} - \boldsymbol{A}_2\|_{\mathrm{F}}^2 &= \operatorname{tr} \left\{ (\boldsymbol{X}\boldsymbol{A}_1\boldsymbol{X}^{\mathrm{T}} - \boldsymbol{A}_2)(\boldsymbol{X}\boldsymbol{A}_1\boldsymbol{X}^{\mathrm{T}} - \boldsymbol{A}_2)^{\mathrm{T}} \right\} \\ &= \operatorname{tr} \left\{ (\boldsymbol{X}\boldsymbol{A}_1\boldsymbol{X}^{\mathrm{T}} - \boldsymbol{A}_2)(\boldsymbol{X}\boldsymbol{A}_1^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}} - \boldsymbol{A}_2^{\mathrm{T}}) \right\} \\ &= \operatorname{tr} \left\{ \boldsymbol{X}\boldsymbol{A}_1\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{A}_1^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}} - \boldsymbol{X}\boldsymbol{A}_1\boldsymbol{X}^{\mathrm{T}}\boldsymbol{A}_2^{\mathrm{T}} - \boldsymbol{A}_2\boldsymbol{X}\boldsymbol{A}_1^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}} + \boldsymbol{A}_2\boldsymbol{A}_2^{\mathrm{T}} \right\} \\ &= \operatorname{tr} \left\{ \boldsymbol{X}\boldsymbol{A}_1\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{A}_1^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}} \right\} - 2\operatorname{tr} \left\{ \boldsymbol{A}_2\boldsymbol{X}\boldsymbol{A}_1^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}} \right\} + \operatorname{tr} \left\{ \boldsymbol{A}_2\boldsymbol{A}_2^{\mathrm{T}} \right\}. \end{aligned} \tag{2.27}$$

In contrast to the balanced problem, the first term in the last row is no longer a constant, since $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ is *not* the identity matrix. In the scope of this work it remained unclear whether there is another way of mapping the traditional graph matching objective problem on a QAP in case of imbalanced problems. The only soft-assign related publication that deals with imbalanced graph matching problems is [7], where Gold and Rangarajan used an alternative definition (see section 1.2.3), whereby graph matching is defined as QAP. Since this seems to be the only way of approximately solving rectangular graph matching problems with the graduated assignment algorithm, we assume the QAP-based definition in the case of unequally-sized sets for the remainder of this work.

The question remains, whether the imbalanced QAPs can be solved using methods for standard QAPs. For their graduated assignment algorithm Gold and Rangarajan [7] introduced slack variables to transform the inequality constraints of the imbalanced problem into equality constraints by adding an additional column to the assignment matrix[6]. No other obvious changes to the algorithm were made, only the summation ranges were adjusted accordingly. Particularly, the slack column was iteratively normalized to 1. As we shall see below, this procedure does not perform as intended, at least from a theoretical point of view.

To obtain an algorithm for that the theoretical properties of the soft-assign quadratic assignment algorithm still hold, we transform the rectangular problem into a standard problem of the size $n \times n$. This is achieved by extending the match matrix by $n - m$ slack columns forming a permutation matrix $\hat{\boldsymbol{X}} \in \tilde{\Pi}_n$. We furthermore extend the cost matrix $\boldsymbol{D}$ to form a $n \times n$ matrix $\hat{\boldsymbol{D}}$, in which new entries are filled with 0. The resulting quadratic

[6]In fact, Gold and Rangarajan proposed to add a slack row as well as a slack column, but the slack row has a different function. Section 2.4.2 will discuss this case.

assignment problem is equivalent to the imbalanced problem, as we can see from

$$\begin{aligned}\operatorname{tr}\left\{\boldsymbol{F}\boldsymbol{X}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\right\} &= \sum_{i=1}^{n}(\boldsymbol{F}\boldsymbol{X}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}})_{ii} = \sum_{i,j=1}^{n}\sum_{k,l=1}^{m} F_{ij}X_{jk}D_{lk}X_{jl} \\ &= \sum_{i,j=1}^{n}\sum_{k,l=1}^{n} F_{ij}\hat{X}_{jk}\hat{D}_{lk}\hat{X}_{jl} = \sum_{i=1}^{n}(\boldsymbol{F}\hat{\boldsymbol{X}}\hat{\boldsymbol{D}}^{\mathrm{T}}\hat{\boldsymbol{X}}^{\mathrm{T}})_{ii} = \operatorname{tr}\left\{\boldsymbol{F}\hat{\boldsymbol{X}}\hat{\boldsymbol{D}}^{\mathrm{T}}\hat{\boldsymbol{X}}^{\mathrm{T}}\right\}.\end{aligned} \tag{2.28}$$

As the extended cost matrix $\hat{\boldsymbol{D}}$ does not add any cost for the slack columns, we can immediately deduce that

$$\min_{\boldsymbol{X}\in\tilde{\Pi}_{n,m}} \operatorname{tr}\left\{\boldsymbol{F}\boldsymbol{X}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\right\} = \min_{\hat{\boldsymbol{X}}\in\hat{\Pi}_{n}} \operatorname{tr}\left\{\boldsymbol{F}\hat{\boldsymbol{X}}\hat{\boldsymbol{D}}^{\mathrm{T}}\hat{\boldsymbol{X}}^{\mathrm{T}}\right\}, \tag{2.29}$$

i. e., the problems are equivalent. In summary, quadratic assignment problems with unequally-sized sets can be transformed into standard quadratic assignment problems in a quite intuitive way without any disadvantages regarding the quality of the solution.

If the graduated assignment algorithm is used to solve such a transformed problem approximately, we can make further simplifications, which mainly influence the running time. At every relaxation step of the discrete-time system, an initial assignment matrix $M^{(r,0)}$ is calculated before the Sinkhorn balancing is applied (see algorithm 2.1, lines 7f). We can easily see that the entries of $\boldsymbol{M}^{(r,0)}$ that correspond to the slack variables are all set to 1. During the following Sinkhorn balancing procedure only row and column sums influence the result. Therefore, we can coalesce the slack columns into only one slack column, which must be iteratively normalized to $n - m$, the former number of slack columns. At this point, the method proposed in [7] and the one proposed here can be discerned: The former normalizes the slack column to 1, whereas a normalization to $n - m$ yields the desired result. Section 4.2.2 gives an experimental comparison between both methods. In summary, the proposed procedure using the coalesced slack columns yields exactly the same assignment as the procedure using all slack columns, but saves the space and the computation time for $n - m - 1$ columns.

2.4.2 Additional slack variables

Gold and Rangarajan [7] proposed to add slack variables as an extra row and column to "handle outliers in a statistically robust manner". In analogy to what was done in section 2.4.1, the assignment matrix and the cost matrices can be extended by one row and one column. We recall that this procedure will not modify the value of the objective function, because the slack variables introduce no further cost. Moreover, the global minimum of the objective will remain unchanged as well. Nevertheless, the slack variables might influence the behavior of the soft-assign QAP algorithm, because the admissible domain is enlarged. This might have advantages as well as disadvantages: The slightly enhanced domain may help the algorithm to circumvent poor local minima; on the other hand, this may cause the algorithm to get stuck in a non-admissible local minimum. Section 4.3 experimentally investigates the influence of the additional slack variables on the performance of the soft-assign quadratic assignment algorithm.

2.5 Convergence Properties

In this section we will study the convergence properties of the preceding algorithm. We will show that the objective function is a discrete-time Lyapunov function for the soft-assign quadratic assignment algorithm. This is done for two cases: First, the existence is shown under the assumption that the Sinkhorn balancing procedure returns an exactly doubly stochastic matrix. As already mentioned, Sinkhorn balancing was shown to converge to a doubly stochastic matrix. But the finite accuracy of machine calculations can only provide an approximately doubly stochastic matrix. Therefore, we will study the theoretical properties under the assumption of approximate convergence of the Sinkhorn balancing procedure as well. It is important to note that the existence of a discrete-time Lyapunov function itself does not guarantee that the *state of the algorithm* converges. Nevertheless, Rangarajan et al. [26] noted that it should be possible to prove convergence of the state under mild assumptions including a finite number of fixed points. Koiran [13] drew similar conclusions for the related case of discrete-time, continuous state Hopfield networks. The hope is, that local convergence in combination with deterministic annealing will lead to convergence into a good local minimum, which is reasonably close to the global minimum of the QAP objective function.

Both parts below follow more or less closely the analysis by Rangarajan et al. [26]. We will reformulate some parts of the proofs to make them easier comprehensible and we will remove some minor flaws.

2.5.1 Exact convergence of Sinkhorn balancing

In the following study of the convergence properties, we assume that the inner Sinkhorn balancing procedure of algorithm 2.1 returns an exactly doubly stochastic matrix. In [26] the following theorem was proven using generic assumptions on the barrier function. This, however, turned out to be difficult, if not impossible, for the case of approximate convergence in the next section. Thus, the proof here uses the negative entropy barrier function for reasons of simplicity.

Theorem 2.5 (Rangarajan et al.)
At each inverse temperature $\beta > 0$, the function

$$F(\boldsymbol{M}) = \frac{1}{2} \sum_{i,j,k,l} C^{(\gamma)}_{ij;kl} M_{ik} M_{jl} + \sum_{i,j} B_{ij} M_{ij} + \frac{1}{\beta} \sum_{i,j} M_{ij} \log(M_{ij}), \qquad \boldsymbol{M} \in \mathbb{R}^{n\times n}_{+} \tag{2.30}$$

is a discrete-time Lyapunov function for the discrete-time synchronous update scheme

$$M^{(r+1)}_{ik} = \exp\left[-\beta\left(\sum_{j,l} C^{(\gamma)}_{ij;kl} M^{(r)}_{jl} + B_{ik} + \mu^{(r+1)}_i + \nu^{(r+1)}_k\right) - 1\right], \qquad \forall i,k;\ r \geq 0 \tag{2.31}$$

under the conditions that the Lagrange parameter vectors $\boldsymbol{\mu}^{(r+1)}$ and $\boldsymbol{\nu}^{(r+1)}$ are set up, such that $\boldsymbol{M}^{(r+1)}$ is a doubly stochastic matrix, $\boldsymbol{M}^{(r)}$ is doubly stochastic, and γ is chosen, such that $\boldsymbol{C}^{(\gamma)}$ is negative definite.

Proof. The change of the objective function value is

$$\begin{aligned}\Delta F :=& F\left(\boldsymbol{M}^{(r)}\right) - F\left(\boldsymbol{M}^{(r+1)}\right)\\ =&\frac{1}{2}\sum_{i,j,k,l} C^{(\gamma)}_{ij;kl} M^{(r)}_{ik} M^{(r)}_{jl} + \sum_{i,j} B_{ij} M^{(r)}_{ij} + \frac{1}{\beta}\sum_{i,j} M^{(r)}_{ij}\log\left(M^{(r)}_{ij}\right)\\ &-\frac{1}{2}\sum_{i,j,k,l} C^{(\gamma)}_{ij;kl} M^{(r+1)}_{ik} M^{(r+1)}_{jl} - \sum_{i,j} B_{ij} M^{(r+1)}_{ij} - \frac{1}{\beta}\sum_{i,j} M^{(r+1)}_{ij}\log\left(M^{(r+1)}_{ij}\right).\end{aligned} \tag{2.32}$$

With $\boldsymbol{\Delta M} := \boldsymbol{M}^{(r+1)} - \boldsymbol{M}^{(r)}$ and the rule $\frac{1}{2}x^2 - \frac{1}{2}y^2 = -\frac{1}{2}(y-x)^2 - (y-x)x$ we obtain

$$\begin{aligned}\Delta F =& -\frac{1}{2}\sum_{i,j,k,l} C^{(\gamma)}_{ij;kl}\Delta M_{ik}\Delta M_{jl} - \sum_{i,j,k,l} C^{(\gamma)}_{ij;kl}\Delta M_{ik} M^{(r)}_{jl} - \sum_{i,j} B_{ij}\Delta M_{ij}\\ &+\frac{1}{\beta}\sum_{i,j} M^{(r)}_{ij}\log\left(M^{(r)}_{ij}\right) - \frac{1}{\beta}\sum_{i,j} M^{(r+1)}_{ij}\log\left(M^{(r+1)}_{ij}\right).\end{aligned} \tag{2.33}$$

If a function $f(x)$ is convex on $\mathbb{R}$, then $f(y) - f(x) \geq f'(x)(y-x)$. Using this property for $f(x) = x\log(x)$ gives

$$\begin{aligned}\Delta F \geq& -\frac{1}{2}\sum_{i,j,k,l} C^{(\gamma)}_{ij;kl}\Delta M_{ik}\Delta M_{jl} - \sum_{i,j,k,l} C^{(\gamma)}_{ij;kl}\Delta M_{ik} M^{(r)}_{jl} - \sum_{i,j} B_{ij}\Delta M_{ij}\\ &-\frac{1}{\beta}\sum_{i,j}\left[\log\left(M^{(r+1)}_{ij}\right) + 1\right]\Delta M_{ij}.\end{aligned} \tag{2.34}$$

Let us now substitute the iteration scheme from equation (2.31) into inequality (2.34):

$$\begin{aligned}\Delta F \geq& -\frac{1}{2}\sum_{i,j,k,l} C^{(\gamma)}_{ij;kl}\Delta M_{ik}\Delta M_{jl} - \sum_{i,j,k,l} C^{(\gamma)}_{ij;kl}\Delta M_{ik} M^{(r)}_{jl} - \sum_{i,j} B_{ij}\Delta M_{ij}\\ &+\sum_{i,k}\left[\left(\sum_{j,l} C^{(\gamma)}_{ij;kl} M^{(r)}_{jl} + B_{ik} + \mu^{(r+1)}_{i} + \nu^{(r+1)}_{k}\right)\Delta M_{ik}\right]\\ =& -\frac{1}{2}\sum_{i,j,k,l} C^{(\gamma)}_{ij;kl}\Delta M_{ik}\Delta M_{jl} + \underbrace{\sum_{i,j}\mu^{(r+1)}_{i}\Delta M_{ij}}_{=0} + \underbrace{\sum_{i,j}\nu^{(r+1)}_{j}\Delta M_{ij}}_{=0}\end{aligned} \tag{2.35}$$

The second and third term in the last row reduce to zero, because exact constraint satisfaction is assumed at both time steps. Because of the negative definiteness of $\boldsymbol{C}^{(\gamma)}$ we finally obtain

$$\Delta F \geq -\frac{1}{2}\sum_{i,j,k,l} C^{(\gamma)}_{ij;kl}\Delta M_{ik}\Delta M_{jl} \geq 0. \tag{2.36}$$

Furthermore, the objective function is bounded below, because each of its terms is bounded below on the domain of doubly stochastic matrices. □

From the proof, we can conclude the following

Corrolary 2.6
*Theorem 2.5 also holds, if $\boldsymbol{C}^{(\gamma)}$ is negative definite in the subspace of matrices with **rows and columns** summing to zero.*

Proof. The statement can be concluded from inequality (2.36) of the preceding proof as well. $\square$

2.5.2 Approximate convergence of Sinkhorn balancing

The following part discusses the convergence properties under weaker assumptions than the previous. We no longer require the Sinkhorn balancing procedure to return an exactly doubly stochastic matrix. Instead, an upper bound on the deviation from the sum constraints is required.

Theorem 2.7 (Rangarajan et al.)
At each inverse temperature $\beta > 0$, the objective function

$$F(\boldsymbol{M}) = \frac{1}{2}\sum_{i,j,k,l} C^{(\gamma)}_{ij;kl} M_{ik} M_{jl} + \sum_{i,j} B_{ij} M_{ij} + \frac{1}{\beta}\sum_{i,j} M_{ij}\log(M_{ij}), \qquad \boldsymbol{M} \in \mathbb{R}_+^{n\times n} \tag{2.37}$$

is a discrete-time Lyapunov function for the discrete-time synchronous update scheme

$$M^{(r+1)}_{ik} = \exp\left[-\beta\left(\sum_{j,l} C^{(\gamma)}_{ij;kl} M^{(r)}_{jl} + B_{ik} + \mu^{(r+1)}_i + \nu^{(r+1)}_k\right) - 1\right], \qquad \forall i,k;\ r \geq 0 \tag{2.38}$$

under the following conditions:

1. *The column sum constraint is exactly satisfied:* $\sum_i M^{(r+1)}_{ij} = 1, \qquad \forall j;\ r \geq 0.$
2. *The row sum constraint is approximately satisfied:*

   $$|\textstyle\sum_j M^{(r+1)}_{ij} - 1| < \epsilon, \qquad \forall i;\ r \geq 0;\ 0 < \epsilon < 1.$$

3. *The quadratic cost matrix $\boldsymbol{C}^{(\gamma)}$ is strictly negative definite with its largest eigenvalue* $\lambda < 0$.
4. *The convergence criterion at each temperature is chosen such that* $\sqrt{\frac{\sum_{ij}\Delta M^2_{ij}}{n^2}} \geq \delta_{\min}$ *where*

   $$\delta_{\min} \geq 2\sqrt{-\frac{\epsilon\left[\sum_l \max_{i,j,p,k}\left(C^{(\gamma)}_{pj;kl} - C^{(\gamma)}_{ij;kl}\right) + \max_{i,p,k}\left(B_{p,k} - B_{i,k}\right) + \frac{1}{\beta}\log\frac{n-1+\epsilon}{1-\epsilon}\right]}{\lambda n}}.$$

For the proof of theorem 2.7, we will need the following bound on the Lagrange parameter vector $\boldsymbol{\mu}^{(r+1)}$.

Lemma 2.8
With the objective function and the update scheme as in theorem 2.7, it holds at each temperature $\beta > 0$ that for each $r \geq 0$

$$\max_i |\mu_i^{(r+1)}| \leq \mu_{max} := \sum_l \max_{i,j,p,k} \left(C^{(\gamma)}_{pj;kl} - C^{(\gamma)}_{ij;kl} \right) + \max_{i,p,k} \left(B_{p,k} - B_{i,k} \right) + \frac{1}{\beta} \log \frac{n-1+\epsilon}{1-\epsilon} \tag{2.39}$$

under the following conditions:

1. *The column sum constraint is exactly satisfied:* $\sum_i M_{ij}^{(r+1)} = 1, \qquad \forall j;\ r \geq 0.$
2. *The row sum constraint is approximately satisfied:*

$$|\textstyle\sum_j M_{ij}^{(r+1)} - 1| < \epsilon, \qquad \forall i;\ r \geq 0;\ 0 < \epsilon < 1.$$

Proof. See section A.5

Proof of theorem 2.7. The first part of this proof is identical to the one for theorem 2.5. Thus, we omit it here. From inequality (2.35) we know that

$$\Delta F \geq -\frac{1}{2} \sum_{i,j,k,l} C^{(\gamma)}_{ij;kl} \Delta M_{ik} \Delta M_{jl} + \sum_{i,j} \mu_i^{(r+1)} \Delta M_{ij} + \sum_{i,j} \nu_j^{(r+1)} \Delta M_{ij}. \tag{2.40}$$

Under the assumption of exact column sum constraint satisfaction, this reduces to

$$\Delta F \geq -\frac{1}{2} \sum_{i,j,k,l} C^{(\gamma)}_{ij;kl} \Delta M_{ik} \Delta M_{jl} + \sum_{i,j} \mu_i^{(r+1)} \Delta M_{ij}. \tag{2.41}$$

It remains to be shown that ΔF is non-negative. This is straightforward just for the first term; it is positive because of the negative definiteness of $\boldsymbol{C}^{(\gamma)}$. The second term in the above inequality, however, can be positive or negative. In the remaining part of the proof we will show that, given certain conditions, the first term is greater than or equal to the absolute value of the second term, which makes ΔF clearly non-negative.

The idea of the remainder is to use an upper bound on $|\mu_i^{(r+1)}|$ and the upper bound λ on the eigenvalues of $\boldsymbol{C}^{(\gamma)}$. The upper bound of the absolute values of the Lagrange parameters ($\forall i : |\mu_i^{(r+1)}| \leq \mu_{\max}$) from lemma 2.8 is used here:

$$\mu_{\max} = \sum_l \max_{i,j,p,k} \left(C^{(\gamma)}_{pj;kl} - C^{(\gamma)}_{ij;kl} \right) + \max_{i,p,k} \left(B_{p,k} - B_{i,k} \right) + \frac{1}{\beta} \log \frac{n-1+\epsilon}{1-\epsilon} \tag{2.42}$$

From the lower convergence bound $\sqrt{\frac{\sum_{i,j} \Delta M_{ij}^2}{n^2}} \geq \delta_{\min}$, we receive that

$$-\frac{1}{2} \sum_{i,j,k,l} C^{(\gamma)}_{ij;kl} \Delta M_{ik} \Delta M_{jl} \geq -\frac{1}{2} \lambda \sum_{i,j} \Delta M_{ij}^2 \geq -\frac{\lambda n^2 \delta_{\min}^2}{2}. \tag{2.43}$$

Using the row sum convergence criterion $|\sum_j \Delta M_{ij} - 1| < \epsilon$, it follows that

$$\sum_{i,j} \mu_i^{(r+1)} \Delta M_{ij} \geq -\mu_{\max} \sum_i \left| \sum_j M_{ij}^{(r+1)} - \sum_j M_{ij}^{(r)} \right| \geq -2n\epsilon\mu_{\max}. \tag{2.44}$$

We finally obtain

$$\Delta F \geq -\frac{\lambda n^2 \delta_{\min}^2}{2} - 2n\epsilon\mu_{\max} \geq 0 \text{ provided that } \delta_{\min} \geq 2\sqrt{-\frac{\epsilon\mu_{\max}}{\lambda n}}, \tag{2.45}$$

or

$$\delta_{\min} \geq 2\sqrt{-\frac{\epsilon\left[\sum_l \max_{i,j,p,k}\left(C_{pj;kl}^{(\gamma)} - C_{ij;kl}^{(\gamma)}\right) + \max_{i,p,k}\left(B_{p,k} - B_{i,k}\right) + \frac{1}{\beta}\log\frac{n-1+\epsilon}{1-\epsilon}\right]}{\lambda n}}. \tag{2.46}$$

Furthermore, the objective function is bounded below (see proof of theorem 2.5). □

Similar to section 2.5.1, we can add the following

Corrolary 2.9
Theorem 2.8 also holds, if $\boldsymbol{C}^{(\gamma)}$ is negative definite in the subspace of matrices with ***columns*** *summing to zero.*

Proof. If λ is an upper bound for the eigenvalues of $\boldsymbol{C}^{(\gamma)}$ in the subspace of matrices with columns summing to zero, the first inequality of (2.43) is still valid and so the remainder of the proof. □

Chapter 3
Related Approximation Techniques

3.1 Artificial Neural Networks and Spin Models

Before techniques related to the soft-assign QAP algorithm are discussed, this section briefly introduces or recalls the properties of Ising and Potts spin systems, which are the basis for important related artificial neural network methods. Some of the following parts are simplified, but should be sufficient to understand the underlying ideas. Please refer to [23] for a more detailed introduction to artificial neural networks in the context of combinatorial optimization.

3.1.1 Introduction to artificial neural networks

Strongly simplified, *biological neural networks* consist of a possibly large amount of interconnected *neurons*, which interpret the signals from other neurons incoming at the dendrites. The input signals add to each other almost linearly. The output signal of the neuron is strongly non-linear, similar to a binary process, but with a finite continuous transition zone. The output signals are transmitted to connected neurons via axons, the "conductors", to the *synapses*, which transmit the signal to the dendrites of the connected neurons. The magnitude of the transmitted signal depends on the strength of the synapse, which can grow or weaken during learning.

For *artificial neural networks* a simple mathematical model is constructed in analogy to biological neural networks. The neurons are modeled as a set of bounded real values, e. g., $\{v_i \in [-1 : 1] \mid i = 1, \ldots, n\}$. Neural models usually employ the following update rule[1]:

$$v_i = g\left(\sum_j W_{ij} v_j - \theta_j\right), \qquad \forall i, \tag{3.1}$$

where the weights W_{ij} (corresponding to the synapses) are non-zero, if the neurons v_i and v_j are connected. The θ_j parameters form thresholds that determine the signal levels of the transition zones. The non-linear transfer function g is usually a sigmoidal, smooth variant of the step function, e. g., $g(x) = \tanh\left(\frac{x}{c}\right)$. A parameter c is usually added to control the steepness of the transfer function in the transition zone.

For the remainder of this chapter, we only consider neural networks of the feedback type, i. e., neurons may feed back to themselves via other neurons. This implies that the system may need many steps of the above update rule to converge to a stable state, or it may not converge at all. In the following section we shall see why and how artificial neural networks can be used to solve combinatorial optimization problems.

3.1.2 Ising spin model

Magnetic systems show striking similarities to artificial neural networks, as we shall see in this section. Moreover, we shall understand that combinatorial optimization problems can be interpreted as the energy minimization of a magnetic system. A magnetic system can be described using the simple *Ising model*, which assumes a large number of binary spins $s_i \in \{-1, 1\},\ i = 1, \dots, n$, which describe the magnetization direction of each atom. It is assumed that each pair of spins affects one another by a constant strength. With that, we can define the following energy function for the Ising model:

$$E(\boldsymbol{s}) = -\frac{J}{2} \sum_{i,j} s_i s_j \tag{3.2}$$

This function basically expresses that equally oriented spins lower the energy of the system. This spin system can be generalized to the *spin glass* model, which has pair-dependent, symmetric couplings between the spins. In this case the energy function can be defined as

$$E(\boldsymbol{s}) = -\frac{1}{2} \sum_{i,j} W_{ij} s_i s_j. \tag{3.3}$$

When looking at the preceding equation, we notice the similarity to combinatorial objective functions, e. g., to the quadratic assignment problem. Just like these objective functions, the spin glass system (with possible negative couplings) has one or many lowest energy states. Thus, bringing a magnetic system into its lowest energy state can be regarded as minimization of a combinatorial objective. Nevertheless, we still have no means of finding these lowest energy states.

The idea is now to embed the magnetic system in a thermal environment. Then, fluctuations of the spins appear according to the Gibbs distribution on the spin states

$$P(\boldsymbol{s}) = \frac{\exp(-E(\boldsymbol{s})/T)}{Z} \tag{3.4}$$

[1] In nothing else is denoted, summation ranges are assumed to be $\{1, \dots, n\}$.

where $Z = \sum_{\hat{s}\in\{0,1\}^n} \exp(-E(\hat{s})/T)$ and T is the temperature[2]. The most probable state is the one with the lowest energy [15], hence finding the most probable state is equivalent to minimizing the energy function. The probability distribution is used to compute the expectation values v_i of the single spins s_i; these are often called *mean fields*. Unfortunately, the expectation values cannot be computed explicitly, but they can be approximated. This may be done using a saddle-point approximation resulting in the mean-field equations. We omit their tedious derivation and refer to [15] or [23]. The mean-field equations of the spin glass model based on a Gibbs distribution are

$$v_i = \tanh\left(-\frac{1}{T}\cdot\frac{\partial E(\boldsymbol{v})}{\partial v_i}\right) = \tanh\left(\frac{1}{T}\sum_j W_{ij}v_j\right), \qquad \forall i. \tag{3.5}$$

At this point we observe the relation to artificial neural networks: The tanh function is the neural transfer function; its argument can be identified as part of the update rule (3.1).

Thus, by using a neural network, we can approximately find the lowest energy state of a magnetic system at temperature T, and by that approximately solve combinatorial optimization problems.

3.1.3 Potts spin model

For approximately solving combinatorial optimization problems with artificial neural networks, it is sometime advantageous to have a network satisfying sum constraints automatically. This is achieved by allowing each neuron to consist of a group of values, of which only one is 1, while the others are confined to 0. The analog spin model is the *Potts spin model*, which uses spin vectors $s = (s_1, \ldots, s_n)^{\mathrm{T}}$ with s being a principal unit vector[3]. The group of n spin vectors is denoted as matrix $\boldsymbol{S}$. If we want to encode row sum constraints into the neural network, we may define the energy function [22][4]

$$E(\boldsymbol{S}) = -\frac{1}{2}\sum_{i,j,k} W_{ij}S_{ik}S_{jk}. \tag{3.6}$$

If the spin model is again embedded in a thermal environment as in the previous section and mean field equations based on the Gibbs distribution are derived, one obtains [11]:

$$U_{ik} = -\frac{1}{T}\sum_j W_{ij}V_{jk}, \qquad \forall i,k \tag{3.7a}$$

$$V_{ik} = \frac{\exp(U_{ik})}{\sum_j \exp(U_{ij})}, \qquad \forall i,k. \tag{3.7b}$$

The V_{ik} are the estimated expectation values of the Potts spins. From equation (3.7b), we observe that $\sum_k V_{ik} = 1$, i. e., the row sums of matrix $\boldsymbol{V}$ are one. Along with the positivity

[2]The term Z is sometimes called partition function.

[3]Principal unit vectors are real vectors, of which one component is 1, the others are 0.

[4]We should note that this energy function has no column constraints.

of the V_{ik} (due to the exponential function) this may be seen as the continuous analog to binary sum constraints.

Compared with the spin glass model, the Potts spin model has not only the advantage that the configuration space is considerably reduced, but also that implementations of this model are far more insensitive regarding parameter choice [22].

3.2 Related Neural Network Models

3.2.1 Hopfield-Tank model

The usage of continuous optimization networks for combinatorial optimization problems was first proposed by Hopfield and Tank in 1985 [see 23]. Since then, neural network methods for optimization have gained considerable interest. Their inherent parallelism makes these algorithms easily implementable and fast on modern parallel architectures. Hopfield and Tank formulated a discrete energy function for the TSP problem similar to the one described in section 1.3.1:

$$E_{\text{obj}}(\boldsymbol{M}) := \sum_{i,j,k} M_{ki} M_{(k+1) \bmod n, j} D_{ij} \tag{3.8}$$

where $\boldsymbol{D}$ is the distance matrix between the cities. This function is to be minimized on the domain of permutation matrices. In principle, the energy function can be extended to arbitrary quadratic assignment problems.

The Hopfield and Tank approach uses the spin glass model to map the combinatorial problem on a neural network. For that, the integrality constraint of permutation matrices is relaxed into a positivity constraint leading to the domain of doubly stochastic matrices. The sum constraints, which bound the domain, are enforced in a "soft" way using penalty functions; the positivity constraint is enforced using a barrier function Φ, whose relation to neural network approaches, in particular to the transfer function, will be clarified below. The Hopfield-Tank energy function can be written as [24]

$$E(\boldsymbol{M}) := E_{\text{obj}}(\boldsymbol{M}) + \frac{A}{2}\sum_{i,j}\sum_{k\neq j} M_{ji}M_{ki} + \frac{B}{2}\sum_{i,j}\sum_{k\neq j} M_{ij}M_{ik} + \frac{C}{2}\left(\sum_{i,j} M_{ij} - n\right)^2 + \frac{1}{\beta}\sum_{i,j}\Phi(M_{ij}). \tag{3.9}$$

The barrier function is given by

$$\Phi(M_{ij}) = M_{ij}\log(M_{ij}) + (1 - M_{ij})\log(1 - M_{ij}). \tag{3.10}$$

The three terms of $E(\boldsymbol{M})$ beginning with the second favor doubly stochastic matrices; their parameters A, B, and C are set to fixed values. The barrier parameter β is also fixed, i. e., no deterministic annealing is performed. It proved in practice that it is impossible to find a parameter set that guarantees convergence to a permutation matrix [24].

Applying the mean-field approximation to the energy (while disregarding the barrier function) leads to weights of the neural network and a neural transfer function:

$$U_{ik} = \sum_j M_{(i+1) \bmod n,j} D_{kj} + A \sum_{j \neq i} M_{jk} + B \sum_{j \neq k} M_{ij} + C \left(\sum_{j,l} M_{jl} - n \right), \qquad \forall i,k \tag{3.11a}$$

$$M_{ik} = g(U_{ik}) = \frac{1}{1 + \exp(-U_{ik}/T)}, \qquad \forall i,k \tag{3.11b}$$

The reason for the transfer function being different from the one in section 3.1.2, is a deviating underlying probability distribution.

The Hopfield-Tank model may lead to states, in which the configuration does not fulfill the doubly stochastic constraints and thus induces a configuration space with redundancies. The alternative energy function

$$E(\boldsymbol{M}) := E_{\text{obj}} + \frac{A}{2} \sum_i \left(\sum_j M_{ij} - 1 \right)^2 + \frac{B}{2} \sum_j \left(\sum_i M_{ij} - 1 \right)^2 + \frac{1}{\beta} \sum_{i,j} \Phi(M_{ij}) \tag{3.12}$$

where row and column constraints are explicitly encoded did not perform substantially better than the original Hopfield-Tank energy [24].

3.2.2 Peterson-Söderberg model

Peterson and Söderberg [22] improved the performance of neural network algorithms for combinatorial optimization by encoding each of the n assignments with only one neuron based on the Potts glass model, in which each spin is allowed to take n different states. With that, one type of sum constraints can be automatically satisfied, which we already referred to as hard constraint satisfaction. We assume a discrete energy function E_{obj} to be given, e. g., by a QAP, and formulate the Peterson-Söderberg energy function as[5]

$$E(\boldsymbol{M}) := E_{\text{obj}} + \frac{A}{2} \sum_j \left(\sum_i M_{ij} - 1 \right)^2 + \frac{1}{\beta} \sum_{i,j} \Phi(M_{ij}) \tag{3.13}$$

where we omit the automatically satisfied row sum constraints.

For now, we suppose that a QAP objective function is given and assume an underlying Gibbs probability distribution (see equation (3.4)). Deriving the Potts spin mean-field equations, while disregarding the barrier function, yields network weights and a neural transfer function [11]:

$$U_{ik} = -\frac{1}{T} \left[\sum_{j,l} C_{ij;kl} V_{jl} + B_{ik} + A \left(\sum_j V_{jk} - 1 \right) \right], \qquad \forall i,k \tag{3.14a}$$

$$V_{ik} = \frac{\exp(U_{ik})}{\sum_j \exp(U_{ij})}, \qquad \forall i,k \tag{3.14b}$$

As we have already seen in section 3.1.3, one type of sum constraints is automatically satisfied for the Potts spin mean-field equations. Thus, the admissible space is much

[5]For simplicity, we ignore the self-amplification term.

smaller compared with the Hopfield-Tank model, which leads to improved results. Peterson and Söderberg [22] furthermore embedded the model in a deterministic annealing framework, which usually improves the solution quality, since at small temperatures the model is closer to the original combinatorial problem.

Let us turn to the relation between barrier function and neural transfer function. Using the Gibbs distribution to derive the mean-field equations results in the transfer function above. We have seen in section 2.1 that for a similar case equations can be derived using a barrier function and Lagrange multipliers. Yuille and Kosowsky [32] explicitly showed that the Peterson-Söderberg neural network model is equivalent to a barrier function approach with a $-x \log x$ barrier function.

When comparing the Peterson-Söderberg model with the soft-assign quadratic assignment algorithm, we observe that the soft-assign QAP algorithm is an extension of this neural network model, in which two types of sum constraints are enforced, namely row sum constraints and column sum constraints. The network weights are identical to those of the Peterson-Söderberg model, besides the penalty term, which is no longer needed. The neural transfer function is not given explicitly, but by an iterative scheme, the Sinkhorn balancing.

3.2.3 Doubly Constrained Network (DCN)

As already mentioned, an algorithm similar to the soft-assign quadratic assignment algorithm was proposed by Ishii and Sato [12], which they originally termed *Doubly Constrained Network (DCN)*. Translated to the variables and objective functions used in chapter 2, they assume a minimization problem as in equation (2.1). Similar to what was done in section 2.1, they derive the following simultaneous equations [cf. 12]:

$$U_{ik} = -\frac{1}{T}\left[\sum_{j,l} C^{(\gamma)}_{ij;kl} V_{jl} + B_{ik}\right], \qquad \forall i,k \tag{3.15a}$$

$$\lambda_k = \sum_i \frac{\exp(U_{ik})}{\sum_j \exp(U_{ij})/\lambda_j}, \qquad \forall k \tag{3.15b}$$

$$V_{ik} = \frac{\exp(U_{ik})/\lambda_k}{\sum_j \exp(U_{ij})/\lambda_j}, \qquad \forall i,k \tag{3.15c}$$

We notice that the equations (3.15b) and (3.15c) scale each row and column sum of $\exp(U_{ik})$ to 1. We see the similarity of these equations to the stationary conditions in section 2.1.5 by looking at the beginning of proof 2.8, especially at equation (A.10).

The DCN algorithm is an iterative scheme based on these simultaneous equations, where the λ_k are obtained by an iterative version of equation (3.15b) as inner loop:

$$\lambda_k^{\text{new}} = \sum_i \frac{\exp(U_{ik})}{\sum_j \exp(U_{ij})/\lambda_j^{\text{old}}} \tag{3.16}$$

This scheme is iterated until approximate convergence and later rescaled such that $\sum_k \lambda_k = 1$, which is possible, since equation (3.15c) is invariant to a scale factor that is applied to all λ_k.

Ishii and Sato [11] have also formulated continuous-time schemes based on differential equations. Experimentally, however, they found these to be inferior to DCN. Furthermore, they recently proposed a promising extension to DCN [10] based on a continuous version of the λopt heuristics[6]. This method was able to improve the best known feasible solutions for two problems of the QAPLIB, but had a significantly longer running time than DCN.

[6]This is an extension of the 2opt heuristics that considers the neighborhood reached by permuting partial permutations of size λ.

Chapter 4
Experimental Analysis of the Soft-assign QAP Algorithm

This chapter documents an experimental analysis of the soft-assign QAP algorithm on various problem instances. Since computer vision applications are the main motivation of this work, graph problems naturally form the largest group of experiments performed. They will be described in sections 4.2 and 4.3. On the other hand, the soft-assign QAP algorithm is very generic and allows to approximate arbitrary quadratic assignment problems. Therefore, the applicability of the algorithm to general QAPs is also assessed. In literature, problems from QAPLIB are often used as benchmark experiments for QAP heuristics, which makes them suitable for analyzing the soft-assign quadratic assignment algorithm experimentally.

4.1 Implementation Notes

To carry out the experiments documented in this chapter the soft-assign QAP algorithm was initially implemented in Mathematica, but this proved to be too slow for extensive tests on larger problems with about 50 nodes. Thus, a reimplementation in a language allowing fast numerical computations became necessary. C++ in conjunction with the *Blitz++* library [30] for array computations was chosen. This library uses advanced C++ template techniques, which provide a high level syntax for array computations that have a performance nearly equivalent to hand-optimized Fortran code [31]. This resulted in a still easily understandable implementation, which was faster than the previous Mathematica code.

Most other parts of the testing facilities, especially non-time-critical sections of them, were implemented in Mathematica using the *Discrete Mathematics* package [29], which provides many useful functions for graphs and graph problems. We had to write code for dealing with weighted graphs, since this package was mainly designed for handling unweighted graphs.

4.2 Large Graph Matching Problems

In this section we report the results from several thousand graph matching experiments conducted on computer-generated graphs with strongly varying properties in the style of the experiments of Gold and Rangarajan [7]. The experiments were carried out on large graphs with mostly $n = 50$ nodes, which makes it intractable to solve the matching problems to optimality, either with brute-force or branch-and-bound methods. To be able to assess the performance of the graduated assignment algorithm on large graphs, a means of comparison is needed, e. g., an estimation of the optimal solution. For that, the conducted experiments used one graph generated randomly and one graph derived from the first according to certain rules. If the methods of derivation are known and the modifications are tracked, we can give an estimate on the best possible matching, which is used to rate the results. Because of this restriction, the purpose of the tests in this section is limited to qualitative analysis of the performance.

The computer-generated graphs are always simple and undirected; both weighted and unweighted graphs are used. Let us first describe the procedure used to generate the random graphs[1]. For every potential undirected edge a given probability and a random number drawn from a uniform distribution on $[0 : 1]$ were used to decide whether this particular edge should be inserted. The above probability will in the following be referred to as *connectivity*, although this method does not guarantee the graph to have this particular connectivity. It is nevertheless used, because generating random graphs with a given connectivity or rather number of edges is much more difficult [29] and unnecessary for our purposes. For the computer-generated weighted graphs, a random edge weight is drawn from a uniform distribution on $[0 : 1]$.

Second, let us describe the rules for deriving the second graph from the first. The structure of the transformation is

$$G_1 \xrightarrow{\pi} G_\mathrm{p} \xrightarrow{D} G_2$$

[1]This procedure was also used for the small, random graphs of section 4.3.

where $\pi \in \Pi_n$ is a permutation and G_1, G_2 are the respective graphs. The transformation D comprises various perturbations, which are applied to reveal the robustness of the algorithm regarding, e. g., the removal of edges or nodes. The particular perturbations will be described together with the experiments below, since they differ for unweighted and weighted graphs. In both cases the modifications of the assignments are tracked. This allows to know the correspondences with the nodes in their original state. We use these correspondences to estimate the best possible matching. Experimentally, it was observed that the estimations are usually good. For testing purposes, 100 small graphs were generated to compare the estimate with the optimal assignment, which was computed using a branch-and-bound algorithm (cf. section 4.3). For about two thirds of the graphs the estimate and the ground-truth were identical; in most other cases a deviation of 1% to 5% was observed. In the average the estimation was about 1.5% worse than the optimum. We can conclude that the estimation is sufficiently good for qualitative tests.

An important comment has to be made on the way of measuring the results of the experiments. In their extensive study on the performance of the graduated assignment algorithm on various types of graphs Gold and Rangarajan [7] used the number of correctly assigned nodes as performance criterion. As they noted, this method has the drawback that it only gives a lower bound on the number of correct matches, since it may ignore optimal matches that do not coincide with the generating permutation. Such a situation occurs, if there are distinct optimal assignments yielding the same objective function value. The experiments conducted for this work use a different method of evaluating the performance; they consider the deviation regarding the objective function, i. e., the objective function value of the obtained assignment is compared to the value of the reference assignment, which may be estimated or computed as described above. This has several advantages: First, it circumvents the above mentioned problem of ignoring optimal matches, because optimal matches always have equal objective function values. Second, assignments with an equal number of correctly assigned nodes may have distinct objective function values. Third, such a measurement makes it also possible to compare the performance on random graph matching problems with the performance on quadratic assignment problems, for which this way of performance measurement is the de facto standard [cf. 3].

The soft-assign quadratic assignment algorithm is controlled by a number of parameters, which are partly critical, as we shall later see. The parameters can be roughly grouped into

- parameters describing the annealing schedule,
- the self-amplification parameter γ,
- convergence and iteration bounds for the inner and outer loop,
- the cleanup method used[2].

The annealing schedule is controlled by three parameters, the initial inverse temperature β_0, the annealing rate β_r, and the final inverse temperature β_f. If the convergence bounds

[2]Strictly speaking, this is no parameter, but the algorithm may have different cleanup methods.

and the iteration bounds are controlled by only one parameter for each loop, there are still 6 parameters to be set up for the experiments. When a graph matching algorithm is used as a component of a more complex computer vision system, selecting parameters by hand is disadvantageous. In that regard, the graph matching experiments in this section use a fixed set of parameters to evaluate the performance. To study the influence of the algorithm parameters on the quality of the solution, further experiments with varying parameters were carried out as described in section 4.2.3. We should note here that for simplicity the theoretical bounds of the convergence criteria derived in section 2.5 are not enforced.

4.2.1 Unweighted graph matching

Unweighted graph matching problems are rarely used in computer vision applications. Since unweighted graph matching problems are usually "harder" to approximate, we still study them here. We discuss them prior to the weighted graph matching experiments, because they are a special cases of weighted graph matching problems. As mentioned in the previous section, the experiments on large, unweighted graphs do not allow computing the best matching, because of the immense time requirements. Therefore, the pair of graphs to be matched is derived from each other giving an estimate on the best solution. For the unweighted graph matching experiments, deleting edges and deleting vertices are the only simple modifications that allow to track the correspondences. Adding edges or vertices is symmetric to deleting edges or vertices from a graph with larger connectivity or more nodes, respectively. In summary, the computer-generated unweighted graphs are governed by the following parameters:

- Size of the first graph
- Connectivity of the first graph
- Percentage of nodes deleted in the second graph
- Percentage of edges deleted in the second graph

For each of these parameters a fixed set of values is used to test the algorithm. The particular composition of these parameter sets will be described below. Testing every possible combination of graph parameters from these sets would be tedious and might also provide an amount of results that is difficult to illustrate and to comprehend. Therefore, only variation of one parameter is considered at a time, while keeping the other three parameters fixed to a default value.

In choosing the parameter sets the intention was to create generic graphs, i. e., the test data should not be restricted to small classes of graphs. On the other hand, the graphs must still have a more or less close relation, because otherwise the estimation of the optimum might not be good. Finally, the following parameter ranges were chosen: The number of nodes in the first graph covers a wide range from just 10 nodes, for which the problems can easily be solved to optimality, up to 150 nodes where even many approximation algorithms have enormous time requirements. The connectivity of the first graph ranges from only 5% probability for a specific edge amounting to very sparse graphs, up to 60%, which induces nearly complete graphs. The percentages of deleted nodes and edges vary between 0% and 60%, the first giving identical graphs (when disregarding the permutation), the latter accounting for large changes in the graphs. Table 4.2 gives an overview about the selected graph parameter sets; the above mentioned default value is emphasized in each set.

Size	$\{10, 20, 30, \mathbf{50}, 75, 100, 150\}$
Connectivity (in %)	$\{5, 10, 15, \mathbf{20}, 30, 40, 60\}$
Nodes deleted (in %)	$\{0, 5, 10, \mathbf{15}, 20, 40, 60\}$
Edges deleted (in %)	$\{0, 5, 10, \mathbf{15}, 20, 40, 60\}$

Table 4.2: Parameter sets for the unweighted graph matching experiments

Instead of generating a random graph for each of the experiments, as it is done, e. g., for the varying connectivity, the first graph is kept fixed for a run using a varying degree of deleted vertices, while the percentage of deleted vertices is *incrementally* increased. This gives an improved coherence of the experimental results. The case of deleted edges is handled analogously.

After having studied the procedure for generating test data, let us turn to the algorithm. All unweighted graph matching experiments use the QAP based definition of graph matching, because most of the instances are rectangular. The edge weight compatibility function f is simply the product of the edge weights. Additional slack variables as discussed in section 2.4.2 were not used, because experiments with them showed that the performance difference is low and that the qualitative behavior is virtually unchanged[3]. The simple binarization proposed in section 2.2.3 is used, which picks one of the largest entries in each column of the continuous assignment matrix to produce a match matrix. This method of binarization is always assumed in this chapter, if not noted otherwise. For simplicity a cleanup by linear assignment was not employed, since the returned assignment matrix was mostly row dominant and close to a match matrix. The 2opt heuristics, however, showed significant improvements (see below) and was thus selected for the experiments. The remaining algorithm parameters were set to: $\beta_0 = 1,\ \beta_r = 1.075,\ \beta_f = 10,\ \gamma = 0,\ \delta = 0.0001,\ r_f = 4,\ \epsilon = 0.001,\ s_f = 30.$

To give a reasonable amount of statistical significance, 100 runs[4] were conducted for each tested parameter quadruple, except for the larger problems with 75, 100, and 150 nodes, for which only 75, 50, and 10 tests were run, respectively. The running time for a single experiment on a current PC (Pentium III, 700 MHz) varied from fractions of a second for 10 node graphs up to 2 hours for some graphs with 150 nodes. For 50 node graphs, the running time was under a minute. As discussed above, the performance is measured by the objective function value; the diagrams show the ratio (in percent) of the objective function value obtained by the soft-assign QAP algorithm to the value

[3]The confidence interval of the mean at a confidence level of 95% was computed (using the t-test) for the results of the experiments with additional slack variables. The interval included the average result of the unmodified algorithm in all cases. Thus the performance change is probably not significant.

[4]New graphs were generated in each run, but with fixed graph parameters.

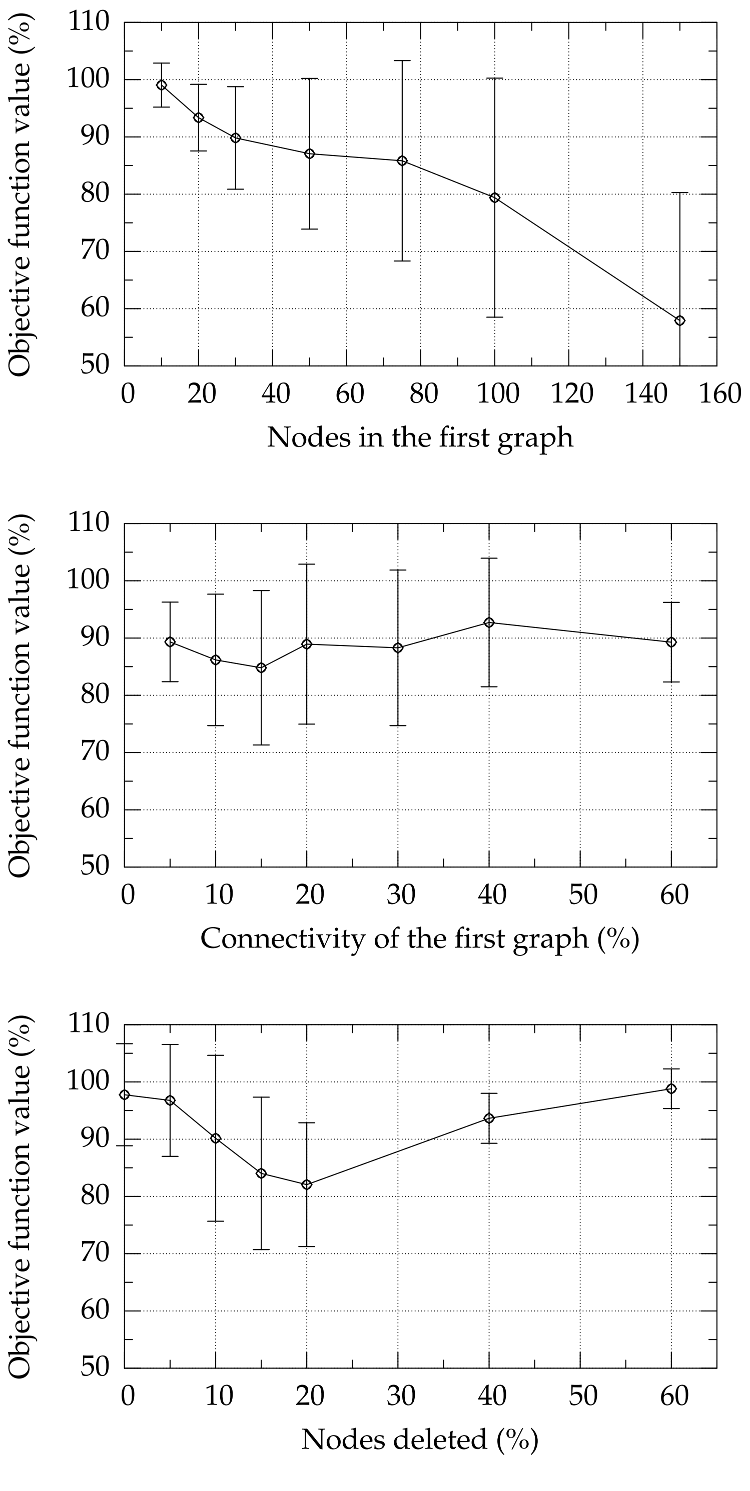


Figure 4.1: Performance on unweighted graph matching problems (I)

estimated using the generating permutation. Figures 4.1 and 4.2 show the results of the 2635 overall experiments. The figures include the mean performance as well as the standard deviation.

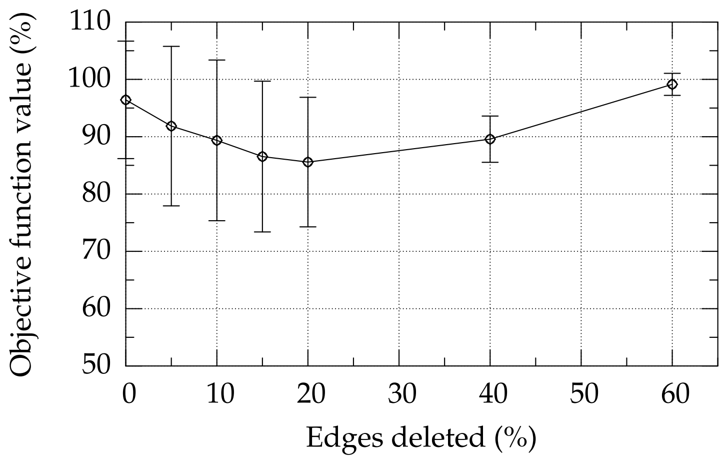


Figure 4.2: Performance on unweighted graph matching problems (II)

Looking at the results, we see that the soft-assign quadratic assignment algorithm suffers from scaling problems. The experiments with 75 or fewer nodes show a mean performance of at least 85% of the estimated optimum, but the larger instances exhibit a strongly decreasing performance. The connectivity of the graphs, however, seems to have little influence on the performance. The varying percentages of deleted vertices and edges reveal a similar profile; the algorithm performs well on slight and large changes, but worse on problems with medium amounts of perturbation. This can be explained by the fact that it might be easier to find a similar subset regarding the nodes or edges of the first graph, if the second graph is considerably smaller regarding the same features. Compared to what Gold and Rangarajan [7] found for a relaxation labeling algorithm, the results obtained here are much better. When comparing the results to those of [7], we notice that the percentage of correctly assigned nodes is considerably worse than the percentage ratio of the objective function values. This is attributable to multiple equivalent matchings of unweighted graphs and to the fact that wrongly assigned nodes may still contribute to the objective function.

Improvement of 2opt greedy strategy

The second set of experiments conducted on unweighted graph matching problems evaluated the 2opt greedy strategy as cleanup heuristics, performed on instances with varying percentage of deleted edges. The algorithm parameters used for these experiments were the same as above. The soft-assign quadratic assignment algorithm with the binarization as proposed in section 2.2.3 delivers the reference result, to which the soft-assign QAP algorithm with an additional 2opt greedy strategy is compared. Figure 4.3 shows the improvement obtained by the 2opt heuristics in percent averaged over 100 runs for each parameter value. The additional marks in the diagram show the confidence in-

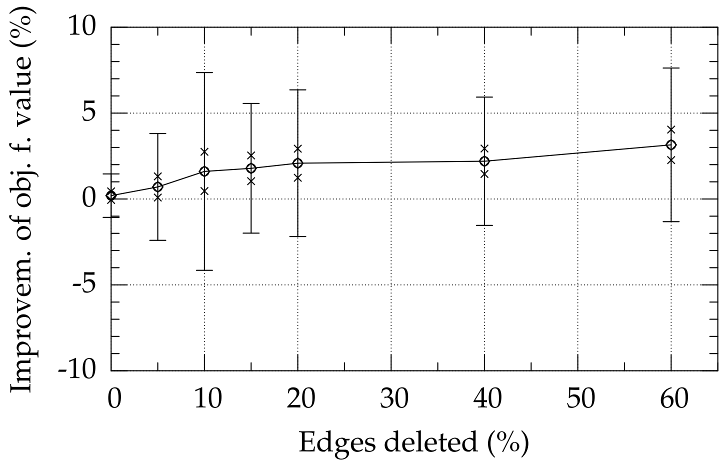


Figure 4.3: Improvement by 2opt heuristics (unweighted graph matching)

terval of the mean at a confidence level of 95%, which was computed using the t-test. We observe that the improvement varies around 2% with the lower confidence bound exceeding 0 for all parameter values except the first; thus, the improvement can be assumed to be significant for most parameter values. The running time for the heuristics was negligible compared to the rest of the algorithm. Because the 2opt heuristics is an effective means of improving the solution quality, it is used for most of the experiments in this chapter.

4.2.2 Weighted graph matching

In addition to the unweighted graph matching experiments, the soft-assign quadratic assignment algorithm was extensively tested on weighted graph matching problems, because they are important for computer vision applications. The experiments were designed as in the previous section, except that edge weights had to be selected and are furthermore candidates for perturbation. The edge weights are, as mentioned above, drawn from a uniform distribution on $[0 : 1]$. They are perturbed with multiplicative noise, which tries to subsume noise of various sources, which in practice influences the determination of the edge weights. As we do not suppose any knowledge on the way of determining the weights, Gaussian noise is assumed to perturb the edge weights. In summary, the following parameters control the test data:

- Size of the first graph
- Connectivity of the first graph
- Percentage of nodes deleted in the second graph
- Percentage of edges deleted in the second graph
- Standard deviation of noise to perturb the edge weights in the second graph

The first four parameters are already known from our experiments on unweighted graph matching problems; thus, we omit their description here. The general test procedure remained unchanged, hence the influence of each parameter for generating the graphs is evaluated separately, while keeping the remaining parameters fixed to a default value.

The graph parameter sets were initially chosen as for the previous experiments. Only an additional parameter set for Gaussian noise was introduced covering 0% to 36% standard deviation. An initial evaluation of these parameter sets revealed that the algorithm performed relatively close to 100% of the expected objective function value for most of the parameter vectors. Figure 4.4 shows the results of experiments with varying degree of deleted edges, conducted as in the previous section besides additional edge noise with 18% standard deviation.

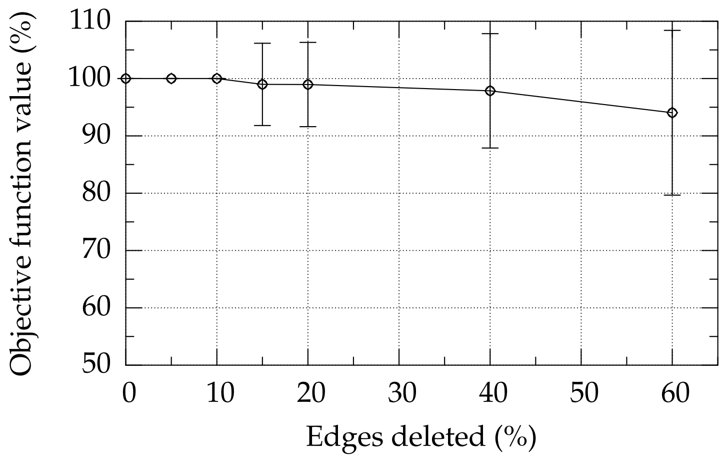


Figure 4.4: Performance on weighted graph matching problems (I)

For the actual weighted graph matching experiments, the parameters were shifted toward more interesting parts with larger variation in the results. Table 4.4 gives an overview about the selected parameter sets; the default value is again emphasized.

Size	$\{10, 20, 30, \mathbf{50}, 75, 100, 150\}$
Connectivity (in %)	$\{4, 8, 12, \mathbf{18}, 25, 35, 50\}$
Nodes deleted (in %)	$\{10, 16, 22, \mathbf{28}, 35, 45, 60\}$
Edges deleted (in %)	$\{10, 16, 22, \mathbf{28}, 35, 45, 60\}$
Standard deviation of noise (in %)	$\{0, 6, 12, \mathbf{18}, 24, 30, 36\}$

Table 4.4: Parameter sets for the weighted graph matching experiments

Let us turn to the algorithm. The experiments on weighted graphs use the alternative graph matching objective, which defines graph matching as QAP, because all problem instances are rectangular. The compatibility function f is sensibly chosen such that fully matching pairs yield a fixed positive value, usually 1, and that all less well matching pairs yield smaller, though positive values. Presuming that the edge weights are uniformly distributed over some interval, Gold and Rangarajan [7] chose f to yield a mean of 0.

When we assume that the weights are drawn from a uniform distribution on $[0:1]$, the compatibility function

$$f(A_{1;kl}, A_{2;ij}) = 1 - 3|A_{1;kl} - A_{2;ij}| \tag{4.1}$$

fulfills all the desired properties, since two values drawn from this distribution are in the average one third apart. It is easily verified that for unweighted, equally-sized graphs with the adjacency matrices $\boldsymbol{A}_1, \boldsymbol{A}_2 \in \{0,1\}^{n\times n}$, the costs of this approach are equivalent to those of the traditional approach.

Regarding the cleanup heuristics, all statements of the previous section apply here as well. Nevertheless, the remaining algorithm parameters were chosen differently to optimize performance: $\beta_0 = 1,\ \beta_r = 1.15,\ \beta_f = 13.3,\ \gamma = 0.75,\ \delta = 0.001,\ r_f = 3,\ \epsilon = 0.01,\ s_f = 10$.

As for the unweighted graph matching experiments, 100 runs were carried out on each tested parameter vector, except for the larger problems with $75, 100$, and 150 nodes, where only $75, 50$, and 10 runs were conducted, respectively. The running time for each experiment was shorter compared to the unweighted graph matching experiments, because of earlier convergence, earlier truncation of the loops, and a larger annealing rate. As before, the results are measured using the objective function value; the diagrams show the ratio of the objective function value obtained by the soft-assign QAP algorithm to the function value estimated using the generating permutation. Figures 4.5 and 4.6 show the results of the 3335 overall tests. When comparing the results to the unweighted graph matching experiments, it has to be considered that the weighted graph matching problems are generally perturbed much stronger.

The results differ in some points from those encountered in the previous section[5]. First, the size of the problem has less influence on the results; even with very large graphs nearly perfect results are observed. The connectivity, in contrary, has a slightly larger influence on the results; the performance at low connectivities is worse than at high connectivities. The results on varying percentages of deleted nodes resemble those of the previous section, whereas the performance on varying degrees of deleted edges has a monotonically decreasing profile for higher percentages of deleted edges. The case of deleted vertices can be explained as in the previous section; the other case is more difficult to interpret. It might be possible that finding a similar subset corresponding to a small second graph is harder, if the edges have weights. We further observe that nearly optimal results are obtained for slight amounts of noise; larger standard deviations exhibit a nearly linearly decreasing performance. Compared to the results of [7], we again notice that the percentage of correctly assigned nodes is considerably worse than the percentage ratio of the objective function values.

[5]When the unweighted graph matching experiments are carried out with the algorithm parameters of this section, the results are worse, but show a profile that is comparable to the original one. Hence, the algorithm parameters do not account for the changed results.

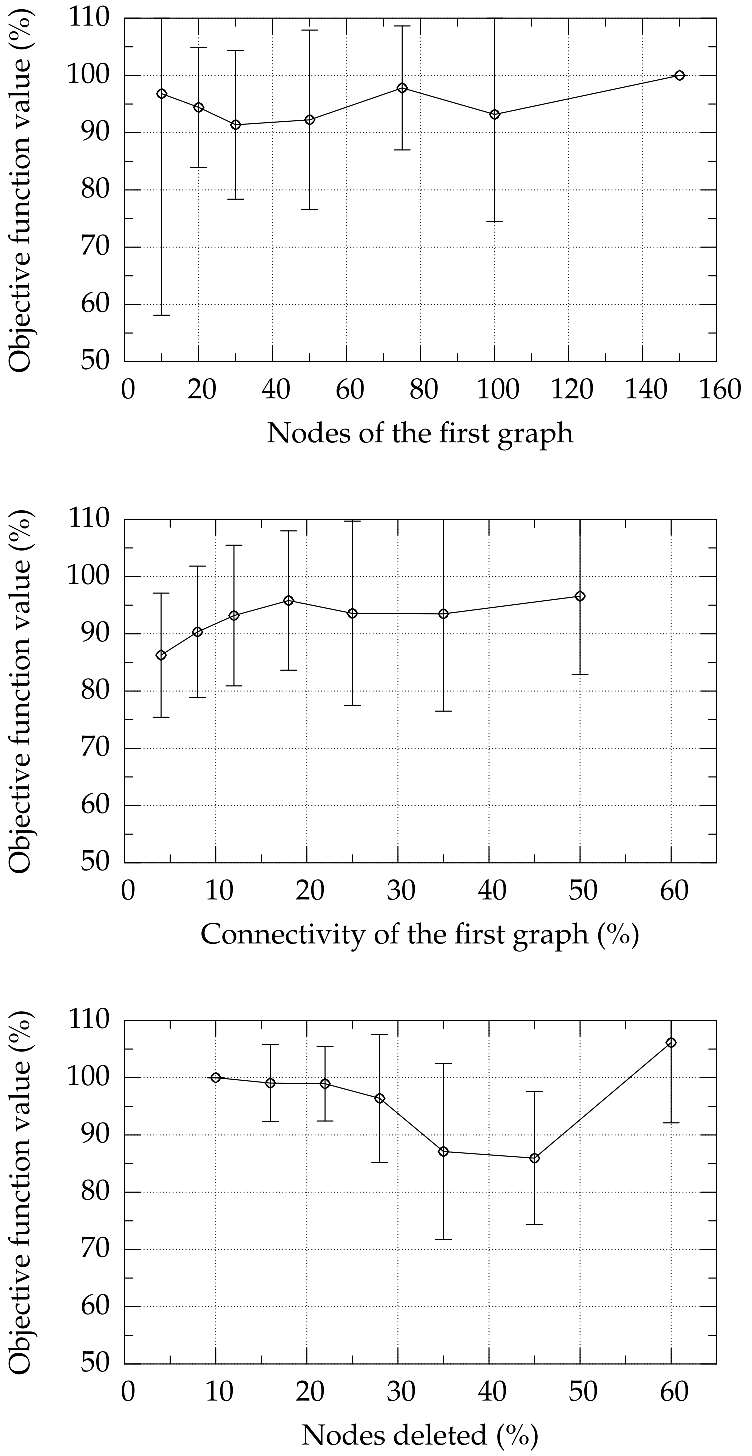


Figure 4.5: Performance on weighted graph matching problems (II)

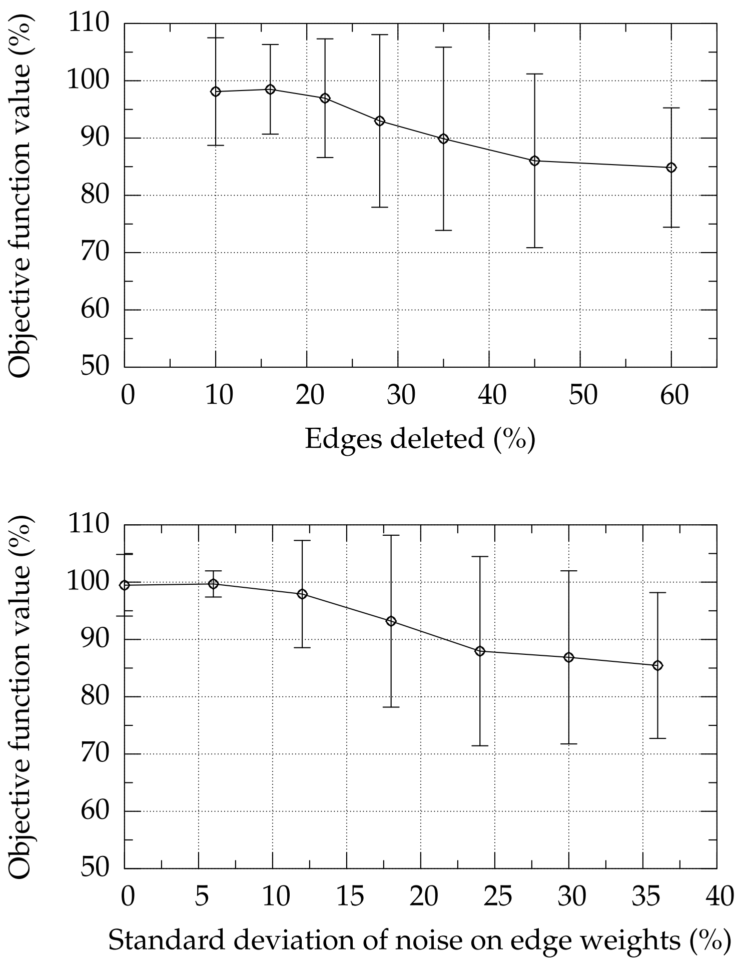


Figure 4.6: Performance on weighted graph matching problems (III)

Improvement of 2opt greedy strategy

As for the unweighted graph matching experiments, the influence of the 2opt heuristics on weighted graph matching problems was studied as well. The test procedure was like the one in section 4.2.1, except that the graph parameters sets of table 4.4 were used. Figure 4.7 shows the results of the overall 700 tests. The additional marks in the diagram show the confidence interval of the mean at a confidence level of 95%. As in section 4.2.1, the performance increase varies around 2%, but shows a larger standard deviation and is only significant for 4 parameter values. This is probably a result of the rougher annealing schedule.

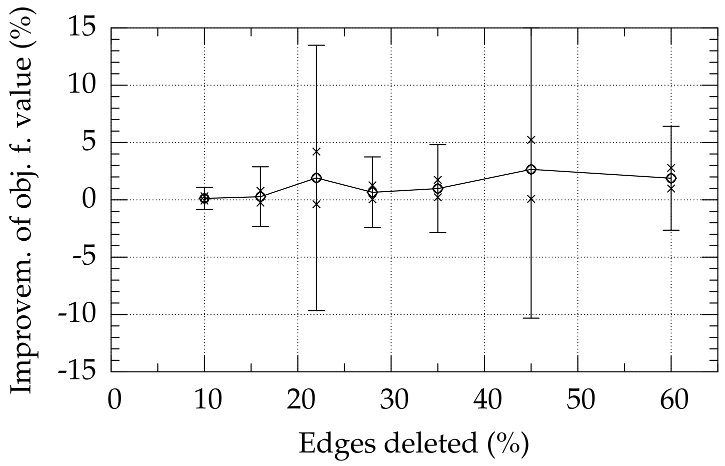


Figure 4.7: Improvement by 2opt heuristics (weighted graph matching)

Influence of slack normalization method

In section 2.4.1, we discussed two alternative ways of handling rectangular graph matching problems. Two implementations were theoretically compared, one using the slack normalization proposed in section 2.4.1, the other using the normalization as proposed by Gold and Rangarajan [7]. To assess their performance difference experimentally, 700 tests with varying degree of deleted edges were conducted. The testing procedure is adapted from the previous weighted graph matching tests. The ratio of the objective function values was measured using the normalization proposed in this work as reference measure. Figure 4.8 shows the results of the experiments. Again, additional marks show the confidence bounds of the mean at 95% confidence. In most cases the change is not significant toward any direction, except for one parameter value where we observe a significant performance deterioration of the normalization proposed by Gold and Rangarajan. We conclude that the normalization proposed in section 2.4.1 is probably also better in practice.

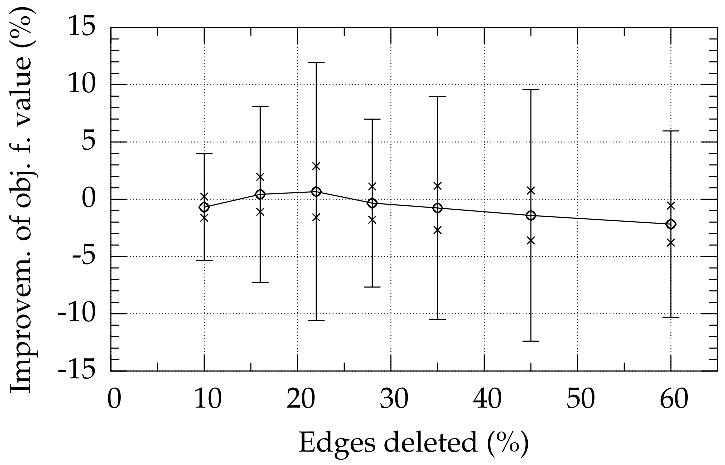


Figure 4.8: Influence of slack normalization method

4.2.3 Influence of algorithm parameters

In the previous section, the conducted experiments used fixed parameter sets for the algorithm to keep the overall parameter space manageable. Nonetheless, experiments with varying algorithm parameters provide useful information, for example, about the robustness of the algorithm regarding its parameters.

The studies with varying algorithm parameters were carried out using weighted graph matching problems with 50 nodes, which were generated using the procedures of section 4.2.2. The parameters for generating the graphs were fixed to the default values of table 4.4.

The algorithm parameters to be tested can be categorized in three groups: the annealing parameters, the self-amplification parameter γ, and the loop bounds. The annealing parameters are the initial inverse temperature β_0, the annealing rate β_r, and the final inverse temperature β_f. To simplify the tests, the convergence and the iteration bounds were controlled by only one scale parameter each; these are r_r and s_r for the outer respective inner loop. The convergence bounds ($\delta = 0.001,\ \epsilon = 0.01$) are scaled with the inverse factor, the iteration bounds ($r_f = 3,\ s_f = 10$) with the unmodified factor. To keep the number of tests and results manageable, only one parameter was varied for a series of experiments, while the other parameters were kept at default values. These were: $\beta_0 = 1,\ \beta_r = 1.15,\ \beta_f = 13.3,\ \gamma = 0,\ r_r = 1,\ s_r = 1$. For each parameter vector 50 runs were conducted. To increase the coherence of the results, only one pair of graphs was generated per run to test the complete parameter range.

As before, the results are measured by the objective function value, but they are normalized to the best result obtained on the parameter range. This normalization is performed, since we are not interested in the absolute result, but in the result relative to other parameter values. Figures 4.9 and 4.10 show the results of overall 4200 tests.

Let us discuss the results. We see that the algorithm shows a nearly constant sensitivity to the initial inverse temperature, as long as it is high enough. We would expect that, since at high temperatures the algorithm might converge into a trivial fixed

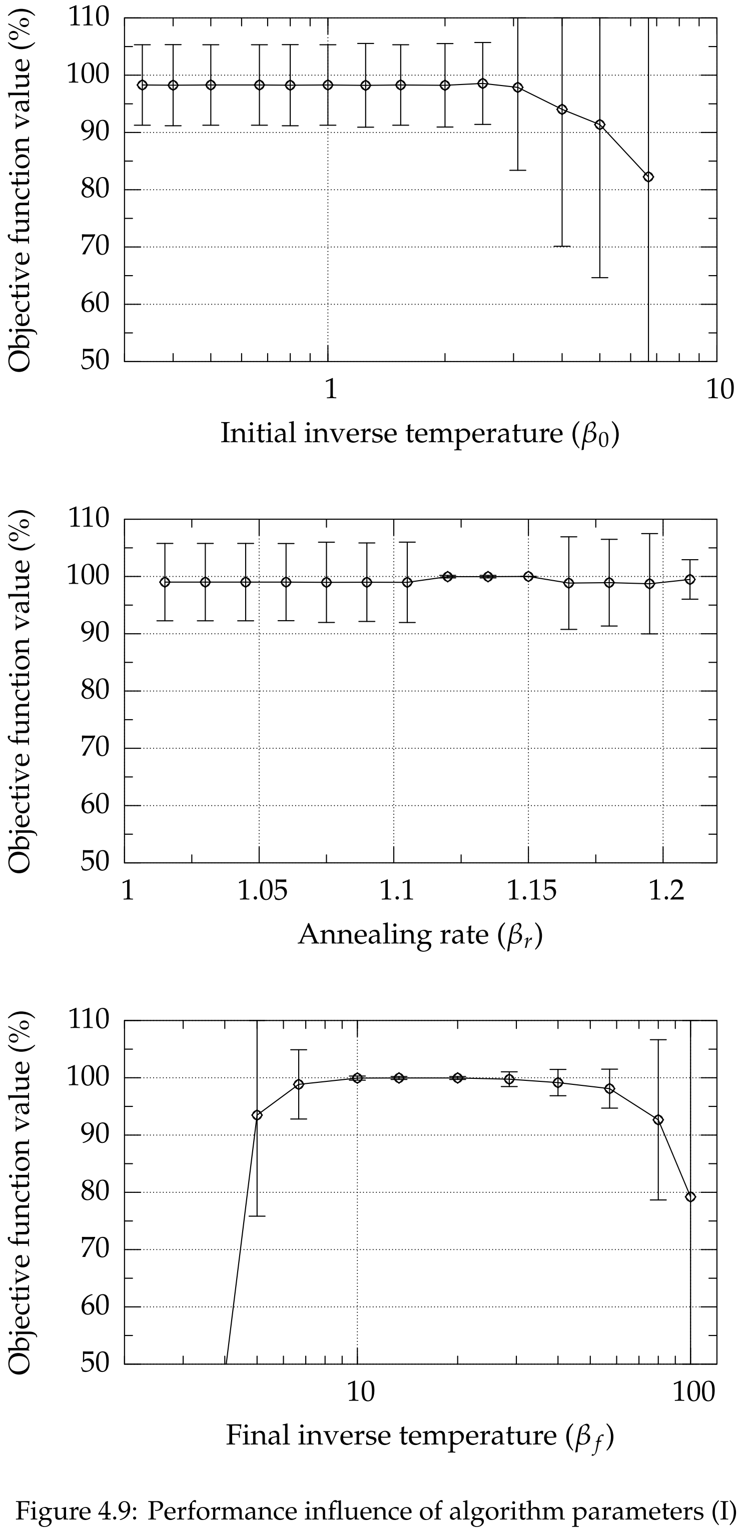


Figure 4.9: Performance influence of algorithm parameters (I)

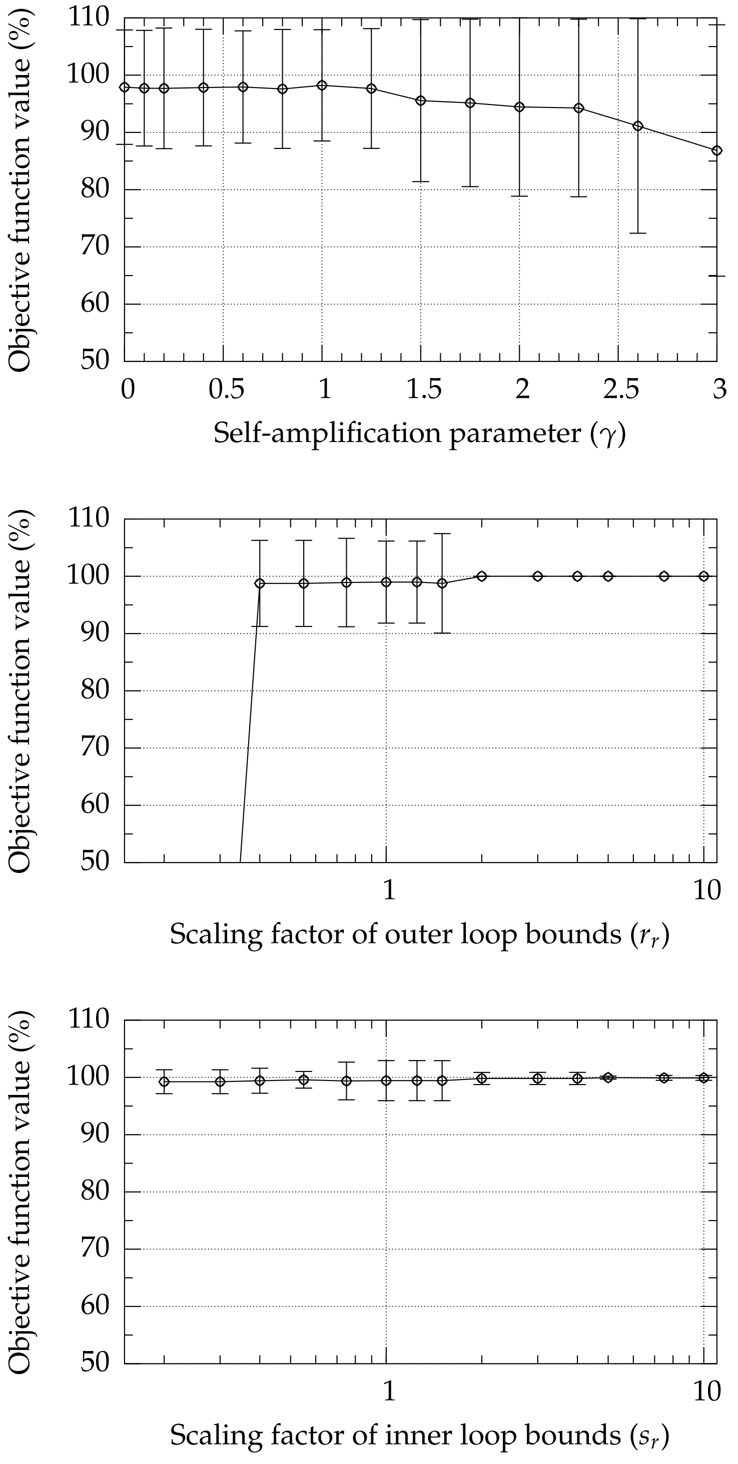


Figure 4.10: Performance influence of algorithm parameters (II)

point (cf. section 2.3.2). Nevertheless, the standard deviation is non-zero, which suggests that a careful setting will improve the results. The non-zero standard deviation can be explained by the finite annealing rate. Variations of the initial temperature cause the temperature steps to lie at different values; hence the algorithm may miss certain, perhaps important bifurcations. The annealing rate shows a range with little variation of the mean and a small standard deviation, which enables us to chose a good value, at least if we assume the graph pairs to have properties similar to the tests here. Ishii and Sato [11] reported for their experiments using the DCN algorithm that a smaller annealing rate always leads to better solutions. This, however, could not be reproduced for the tests conducted here (cf. section 4.4). The final inverse temperature has a limited parameter range, in which the algorithm shows little parameter sensitivity. The deteriorating performance for high values is probably due to the 2opt heuristics, whose success strongly depends on the condition of the pair-exchange neighborhood.

At low values the self-amplification parameter shows little variation of the mean, but a considerable standard deviation. The performance strongly deteriorates for high values. This is surprising, since the theoretical studies of section 2.3.1 gave a lower bound for γ. We infer that the fulfillment of the theoretical bound alone does not guarantee good results. In addition, the large standard deviation suggests that the selection of the self-amplification parameter is crucial to optimal performance.

The results on experiments with varying outer loop scales exhibit nearly no variation of the mean and a nearly vanishing standard deviation, as long as a lower bound is exceeded. That suggests that good values can be found, at least if the properties of the graph pairs are comparable to the experiments here. The inner loop shows little variation in the performance and also a relatively small standard deviation; higher loop scale factors mostly improve the results. In summary, the annealing schedule and the self-amplification parameter are crucial to optimal performance; the optimization of the convergence and iteration bounds is less critical but a good setting may further improve the results.

Although the initialization of the soft-assign QAP algorithm is not directly controlled by a parameter, it nevertheless might influence the results, since it uses random values. To determine the influence, 100 runs were conducted for several computer-generated problems with default parameter sets. Despite the random initialization, the returned objective function value was the same for all runs on each graph pair. We thus conclude that the initialization is uncritical.

4.3 Graph Matching Problems with Known Ground-truth

In this section we will complement our graph matching studies with a series of tests conducted on small, weighted graphs, for which the computation of the optimal solution is still tractable. The size of the instances is therefore limited to about 15 vertices, which was also chosen for the experiments.

In contrary to the weighted graph matching experiments described above, the graphs to be matched can be generated independently of each other, since the optimal solution is computed and not estimated. For that, each graph is generated randomly with

	without slacks/lin. asg./2opt				*with 2opt*			
Connectivity	*worst*	*mean*	*std. dev.*	$o_{\max}$	*worst*	*mean*	*std. dev.*	$o_{\max}$
5%	0.0000	0.9191	0.1287	27	0.7543	0.9739	0.04492	50
10%	0.5594	0.9051	0.07983	5	0.8076	0.9478	0.04728	12
20%	0.0000	0.8981	0.1516	2	0.8196	0.9496	0.03883	6
30%	0.5581	0.9427	0.06240	2	0.8454	0.9585	0.03050	6
40%	0.7520	0.9494	0.04426	6	0.8813	0.9607	0.02792	8
60%	0.7956	0.9560	0.03342	4	0.9102	0.9688	0.01881	6

	with slacks				*with linear assignment*			
Connectivity	*worst*	*mean*	*std. dev.*	$o_{\max}$	*worst*	*mean*	*std. dev.*	$o_{\max}$
5%	0.0000	0.9136	0.1309	27	0.0000	0.9248	0.1264	32
10%	0.6831	0.9142	0.06434	4	0.5594	0.9068	0.07847	5
20%	0.0000	0.9107	0.1261	1	0.0000	0.9134	0.1506	5
30%	0.5023	0.9421	0.06109	3	0.5850	0.9514	0.04858	3
40%	0.6621	0.9453	0.05664	6	0.7520	0.9545	0.03989	6
60%	0.6253	0.9539	0.04370	4	0.8054	0.9634	0.02641	5

	with slacks/lin. asg./2opt				*theor. convergence bounds*			
Connectivity	*worst*	*mean*	*std. dev.*	$o_{\max}$	*worst*	*mean*	*std. dev.*	$o_{\max}$
5%	0.7357	0.9720	0.04872	49	0.0000	0.6643	0.2918	12
10%	0.8076	0.9495	0.04616	11	0.0000	0.2342	0.2302	0
20%	0.8560	0.9494	0.03665	6	0.0000	0.1691	0.1043	0
30%	0.8696	0.9588	0.02855	5	0.08374	0.2810	0.09843	0
40%	0.8940	0.9610	0.02605	7	0.2301	0.3958	0.08769	0
60%	0.9191	0.9684	0.01828	7	0.3358	0.5473	0.07196	0

Table 4.5: Performance on small, random weighted graphs

	with 2opt		*with slacks*		*with lin. assignment*	
Connect.	*lower b.*	*upper b.*	*lower b.*	*upper b.*	*lower b.*	*upper b.*
5%	0.02782	0.08176	−0.04167	0.03073	−0.02986	0.04128
10%	0.02440	0.06105	−0.01115	0.02930	−0.02039	0.02376
20%	0.02054	0.08256	−0.02632	0.05147	−0.02678	0.05749
30%	0.002064	0.02952	−0.01777	0.01667	−0.006906	0.02430
40%	0.001010	0.02167	−0.01824	0.01012	−0.006629	0.01687
60%	0.005266	0.02042	−0.01286	0.008846	−0.0009741	0.01583

	with sl./lin. asg./2opt		*theor. converg. bounds*	
Connect.	*lower b.*	*upper b.*	*lower b.*	*upper b.*
5%	0.02570	0.08014	−0.3179	−0.1917
10%	0.02622	0.06265	−0.7191	−0.6227
20%	0.02039	0.08221	−0.7653	−0.6927
30%	0.002524	0.02966	−0.6847	−0.6387
40%	0.001514	0.02180	−0.5730	−0.5341
60%	0.004948	0.02000	−0.4243	−0.3929

Table 4.6: Confidence Intervals of the mean difference

Connectivity	β_0	β_r	β_f	γ	r_r	s_r
5%	1.0	1.15	20	0.25	2.5	1
10%	1.0	1.15	40	0.25	1	1
20%	1.0	1.15	40	0	2.5	1
30%	1.0	1.075	40	0	1	2.5
40%	1.0	1.075	40	0	1	2.5
60%	0.5	1.075	40	0	2.5	1

Table 4.7: Parameters of the experiments with small, random weighted graphs

the connectivity as its only parameter. This also means that the graphs to be matched only have the connectivity, their size, and the range of the edge weights in common.

For computing the optimal solution, the implementation of a branch-and-bound algorithm was used, which is distributed in conjunction with the data of the QAPLIB [3]. It is based on the well-known Gilmore-Lawler Bound [21, pp. 19f]. Because this implementation is not capable of using floating point edge weights, integer problems were generated with edge weights ranging from 0 to 100, which is no significant limitation. The set of connectivities was taken from the large graph experiments (cf. table 4.4). For each connectivity value, the tests series comprised 100 experiments.

So far, we have not discussed the details regarding the algorithm. The traditional graph matching objective was used for the experiments in this section, because the instances are balanced. Experiments with and without additional slack variables as proposed by Gold and Rangarajan [7] (cf. section 2.4.2) were conducted. The performance impact of the 2opt heuristics was tested as well. In addition, an alternative cleanup heuristics was evaluated, which uses a simple linear assignment problem to binarize the assignment matrix. Subsequently, a series of tests was run using all three techniques, additional slack variables, linear assignment, and the 2opt heuristics. The algorithm parameters were semi-automatically optimized for a small subset of each test series using 48 distinct parameter vectors. The initial temperature was chosen high enough to exceed the critical temperature. Table 4.7 shows the algorithm parameters determined. The inner and outer loop was controlled by only one parameter each. The convergence bounds ($\delta = 0.0001,\ \epsilon = 0.001$) were scaled with the inverse scaling parameter, the iteration bounds ($r_f = 4,\ s_f = 30$) with the scaling factors r_r and s_r themselves. The convergence bounds determined are not equal among the test series; tighter bounds did not always yield better results. A spot check on the instances with 30% connectivity revealed that a scale factor of $r_r = 2.5$ decreased the average performance by approximately 2.5%. We should note that the convergence related parameters of the experiments described above do not necessarily fulfill the theoretical bound derived in section 2.5. Because the focus here is on quantitative results, another series of tests was carried out that used convergence parameters set up to fulfill the theoretical bounds. The parameters were chosen as in [26] ($\epsilon = 10^{-6},\ \lambda = 0.01$).

The results are given in table 4.5, which shows the worst and the mean performance on the data together with the standard deviation. The best performance was 100% for all connectivity values and algorithm variations. In addition, the number of optimal assignments (as determined by an optimal objective function value) found ($o_{\max}$) is shown. Table 4.6 gives the confidence intervals of the mean difference. These intervals were computed at a confidence level of 95% with the unmodified algorithm as reference.

When looking at the results, we notice that each test series of the unmodified algorithm shows satisfactory results with optimized algorithm parameters. Although the parameter optimization was performed only on a small subset of the test data, the parameter values seem to be suitable for nearly the whole series of tests. The measured improvement obtained by the 2opt heuristics is comparable to the previous results, i. e., about 1% to 5% improvement. The confidence intervals again document a significant improvement. Furthermore, we see that additional slack variables induce only a marginal improvement, for some connectivity values even a deterioration. Their use is arguable, if we furthermore consider that the change is not significant (based on the confidence intervals) toward any direction. The linear assignment cleanup technique also shows mixed results; the confidence intervals show no significant change, but a tendency toward a small improvement might be assumed. The combination of all three aforementioned techniques performs similar to the algorithm with the 2opt heuristics alone. The satisfaction of the theoretical convergence bounds yields surprisingly bad results. Their mean performance is significantly worse compared to the tests using fixed bounds. One possible interpretation of these results is that the strict convergence into a local minimum causes the algorithm to get stuck in a bad local minimum. Furthermore, the theoretical bounds lead to an early termination of the outer loop ($\delta \approx 0.02$), which may prevent the algorithm from tracking important bifurcations. Different parameters for the latter experiments were not able to improve the performance considerably.

4.4 QAPLIB Problems

The final part of the experiments uses instances from the well known QAPLIB, a compilation of quadratic assignment problems from various applications. The size of the problems ranges from only 12 facilities up to 256 facilities. For some of the given instances up to 32 facilities the optimal solution is known. In the other cases, the library gives the objective function value of best approximation method as well as the best known lower bound for the solution. Many of the problems in the QAPLIB can be characterized as difficult, because the gap between the best approximate solution and the best found lower bound is often 30% or more.

The experiments were conducted using the soft-algorithm QAP algorithm with additional slack variables, linear assignment as cleanup heuristics, and the 2opt heuristics for local improvement of the solution. The parameters of the algorithm were determined semi-automatically using a fixed set of values for each parameter. The convergence and iteration bounds were again controlled by only one scale parameter each. The values to be scaled were the same as in section 4.3. For problems with $n \leq 40$, 196 different parameter vectors were tested; for $40 < n < 100$ only 6 parameter vectors were employed. Finally, the experiments for very large instances of $n \geq 100$ used only two parameter vectors. For simplicity, the initial temperature is not enforced to exceed the critical temperature.

Table 4.8 shows an excerpt of the results of the experiments. We should note that the size of the problem is encoded into the problem name. If the optimal value is known, it is given in the table. If the optimum has not (yet) been found, the best known approximation value is given and marked with an asterisk (*). For every instance, the best objective function value prior and after the 2opt optimization step is shown. The parameters in table 4.8 are those of the run that yielded the best result after the 2opt greedy strategy was applied. See table A.1 in section A.6 for the results of all except one symmetric QAPLIB instances[6].

To assess the performance difference between separately optimized parameters and a fixed parameter set, some of the experiments were also carried out using the fixed

[6]The instance Tai256c was omitted.

Name	*feas. solution*	*w/o 2opt*	*w/ 2opt*	β_0	β_r	β_f	γ	r_r	s_r
Chr12a	9552	14406	9916	0.5	1.15	40.0	0.25	1	1
Had20	6922	7258	6934	0.5	1.15	20.0	0.25	1	1
Nug30	6124∗	6676	6186	0.5	1.05	20.0	0.25	2.5	1
Esc32a	130∗	152	134	0.5	1.025	20.0	0	1	1
Sko56	34458∗	39616	35200	0.5	1.05	20.0	0.25	1	1
Tai80a	13557864∗	15385348	13990144	0.5	1.05	20.0	0.25	1	1
Wil100	273038∗	290534	274458	0.5	1.05	20.0	0.5	1	1
Tho150	8134030∗	9148466	8275938	0.5	1.05	20.0	0.5	1	1

Table 4.8: Excerpt of QAPLIB results

algorithm parameters $\beta_0 = 0.5$, $\beta_r = 1.075$, $\beta_f = 20$, $\gamma = 0$, $r_r = 1$, $s_r = 1$. Table 4.9 shows the performance of the different runs.

	w/o param. optimization		*w/ param. optimization*	
Name	*w/o 2opt*	*w/ 2opt*	*w/o 2opt*	*w/ 2opt*
Chr12a	22516	10214	14406	9916
Chr12b	18884	10102	13888	9742
Tai12a	256874	241738	245006	230704
Scr15	70788	55448	65632	51140
Chr15a	22884	12152	12900	10976
Chr15b	37762	10800	15776	8640
Chr15c	37712	13556	16528	11352

Table 4.9: Performance with and without parameter optimization

The results of the QAPLIB experiments show that good feasible solutions could be obtained for most of the instances, often deviating less that 5% from the optimum. We observe that the exhaustive parameter search for the small problems did not yield much improved results compared with the parameter search for the medium-sized problems (assuming that they are of comparable difficulty). We furthermore notice that even instances of the same source have distinct optimal parameter vectors. The convergence bounds show a similar behavior as in the last section; tighter bounds do not always improve the result. In addition, smaller annealing steps only sometimes yield better results. When looking at table 4.9, we observe that the parameter optimization strongly improved the results, especially if no 2opt heuristics was applied. For these instances, the performance without parameter optimization was similar to the performance on weighted graph matching problems. When we compare the results presented here with the results on a few QAPLIB problems obtained using the DCN algorithm [11], we observe that the results given here are worse although the algorithm is practically identical. The gap of the objective function value for the QAPLIB instance Tai20a is 3.5% in our case, while it is only 1.3% for the DCN. This is probably a result of inferior algorithm parameters: Ishii and Sato gave some of their parameters used to obtain the results; the parameters strongly vary between the different instances and suggest that Ishii and Sato probably tested large sets of parameter vectors.

Chapter 5
Summary

Object recognition is a central topic of computer vision research. In particular, view-based object representations gained considerable interest. Object views are often expressed as a set of features with relations on them, which can be represented as weighted graph. Thus, matching weighted graphs is a requirement of many view-based object recognition systems, but it is computationally very hard and for large instances intractable on today's computers. Hence, there is a strong interest for approximation algorithms that find suboptimal solutions to these matching problems in polynomial time.

In chapter 1 we formally defined the weighted graph matching problem and saw that it is a special case of the quadratic assignment problem (QAP). We formulated QAPs in several ways and saw that they form a broad class of combinatorial optimization problems, which comprises many classical problems like the traveling salesman problem. Furthermore, we introduced some of the most important classes of approximation algorithms for QAPs.

The soft-assign quadratic assignment algorithm is an approximation algorithm for QAPs, which recently gained significant interest. In chapter 2 we derived this algorithm step-by-step, closely studying every component. After presenting the algorithm, we discussed the influence of some of the algorithm parameters theoretically, namely the self-amplification parameter and the initial inverse temperature. For both lower bounds could be estimated that guarantee certain properties. Subsequently, we studied how rectangular matching problems can be approximately solved using the graduated assignment algorithm. We saw that the rectangular graph matching problem in its traditional definition is no longer a quadratic assignment problem. Furthermore, we observed that the time and memory requirements of the soft-assign QAP algorithm on rectangular QAPs can be reduced without any deterioration of the solution quality. The final section of chapter 2 was devoted to important convergence related properties based on different assumptions regarding the algorithm. The theoretical results suggest that the algorithm converges locally regarding the objective function under certain assumptions.

After briefly introducing artificial neural networks in chapter 3, we discussed related neural network techniques for combinatorial optimization including the Doubly Constrained Network (DCN), which is very similar to the soft-assign QAP algorithm. Furthermore, we described the relation of the soft-assign quadratic assignment algorithm to neural network algorithms.

An experimental analysis of the algorithm using various graph matching and quadratic assignment problems was documented in chapter 4. We introduced methods of generating graphs either randomly or by derivation, followed by a discussion about the measurement of the results. Several thousand graph matching experiments were conducted, both on unweighted and on weighted graphs. They revealed that the soft-assign QAP algorithm shows comparably good solutions on the different instances using fixed algorithm parameters, except for large unweighted graph matching problems. The 2opt cleanup heuristics showed a significant improvement of the results for most of the experiments. Tests of the parameter sensitivity on weighted graph problems revealed mixed results. For these graphs, the soft-assign QAP algorithm is relatively insensitive to some of the parameters, but others are crucial for optimal performance, especially the annealing schedule and the self-amplification parameter. In addition, experiments on small graphs, for which the optimal solution can be computed, were conducted. For these experiments the parameters were tuned on a subset of the test data. The choice of the convergence bounds was observed to be important for optimal performance on these instances. The experiments showed nearly equally good performance on most instances and revealed that minor modifications of the algorithm, namely additional slack variables and cleanup by linear assignment, do not considerably improve the results. Surprisingly, the fulfillment of the theoretical convergence bounds yielded significantly worse performance results.

Finally, experiments were carried out on all symmetric QAPLIB [3] instances, which showed that the gap to the optimal solution is often less than 5%, provided that the algorithm parameters were semi-automatically tuned for every instance. However, a comparison with results on some QAPLIB problems computed using the very similar DCN algorithm suggests that a more thorough parameter choice still yields strongly improved results. In summary, chapter 4 showed that the soft-assign quadratic assignment algorithm needs manual or semi-automatic parameter tuning procedures on hard problem instances to produce good results. All three parameter groups, namely annealing parameters, self-amplification parameter, and convergence and iteration bounds are important for optimal performance, whereby the first two groups showed the largest performance impact. On easier instances fixed parameter settings might be sufficient in practice.

Appendix A
Proofs and Details

A.1 Multigraph Matching

In this section we will introduce a mapping of the *multigraph matching problem* onto a quadratic assignment problem in the general form by Lawler. Multigraphs deviate from standard weighted graphs by allowing each pair of vertices to have multiple links, whereby a weight is associated with every link.

In computer vision applications, *attributed relational graphs* [e.g. 17] are more common, but can be expressed as a slightly enhanced multigraph. Multigraphs are used here, because they are a standard type in graph theory. We will use them with a small generalization; their vertices have an associated weight vector. Let us formalize the multigraphs. We assume that the graphs to be matched, here denoted as $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, each have n vertices ($|V_1| = |V_2| = n$). V_1, V_2 are indexable sets of vertices $\{v_{p;0}, \ldots, v_{p;n}\}$, $p = 1, 2$. Each graph has an associated *vertex weight function* assigning a vector of real weights to each vertex ($w_{\mathrm{v};1} : V_1 \to \mathbb{R}^m$ and $w_{\mathrm{v};2} : V_2 \to \mathbb{R}^m$). We assume that each set of edges E_1, E_2 has a corresponding function $f_1 : E_1 \to \{(u,v) \in V_1 \times V_1 \mid u \neq v\}$ and $f_2 : E_2 \to \{(u,v) \in V_2 \times V_2 \mid u \neq v\}$, which assigns a pair of vertices to each element out of the set of edges. We observe that this is more flexible than the traditional notion of graphs, because multiple links are allowed. Finally, we assign an *edge weight function* to each set of edges: $w_{\mathrm{e};1} : E_1 \to \mathbb{R}_+$ and $w_{\mathrm{e};2} : E_2 \to \mathbb{R}_+$.

Unfortunately, adjacency arrays (with three dimensions) cannot be derived in a straightforward way, because the number of parallel edges is not limited in the above definition. Nevertheless, we can directly define the graph matching problem as a quadratic assignment problem similar to the traditional form (cf. section 1.2.1). The edge weights are, as usual, comprised by the quadratic costs:

$$C_{ij;kl} := \sum_{p\in E_1} \sum_{q\in E_2} w_{\mathrm{e};1}(p) \cdot w_{\mathrm{e};2}(q) \cdot \delta_{f_1(p),(v_{1;k},v_{1,l})} \cdot \delta_{f_2(q),(v_{2;i},v_{2,j})}, \qquad \forall i,j,k,l \tag{A.1}$$

The vertex weights form a linear cost matrix

$$B_{ij} := w_{\mathrm{v};1}(v_{1,j})^{\mathrm{T}} w_{\mathrm{v};2}(v_{2;i}), \qquad \forall i,j. \tag{A.2}$$

By that, multigraphs or attributed relational graphs can be approximately matched using the soft-assign quadratic assignment algorithm.

A.2 Properties of Kronecker Products

The following section gives some basic properties of Kronecker products, which are needed in this work. Proofs of their correctness are omitted here, since [9] contains in-depth descriptions of the properties and their proofs.

For two matrices $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{r\times s}$ the Kronecker product is defined as

$$\boldsymbol{A} \otimes \boldsymbol{B} := \left(A_{ij}\boldsymbol{B}\right)_{i,j=1}^{m,n} \in \mathbb{R}^{mr\times ns}. \tag{A.3}$$

Kronecker products are obviously linear regarding both matrices. Furthermore, it can be shown to be associative and distributive with respect to addition. Naturally, it is *not* commutative.

For some of the following properties, matrices are needed, whose columns are stacked into one large vector. For $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ this is defined as

$$\operatorname{vec} \boldsymbol{A} := (A_{\cdot,i})_{i=1}^{n} \in \mathbb{R}^{mn}. \tag{A.4}$$

The following properties are given in an informal way. They are only valid under intuitively recognizable circumstances, i. e., compatible dimensions or invertibility.

$$\begin{aligned}
(\boldsymbol{A}\otimes\boldsymbol{B})^{\mathrm{T}} &= \boldsymbol{A}^{\mathrm{T}}\otimes\boldsymbol{B}^{\mathrm{T}}\\
(\boldsymbol{A}\otimes\boldsymbol{B})(\boldsymbol{C}\otimes\boldsymbol{D}) &= \boldsymbol{AC}\otimes\boldsymbol{BD}\\
(\boldsymbol{A}\otimes\boldsymbol{B})^{-1} &= \boldsymbol{A}^{-1}\otimes\boldsymbol{B}^{-1}\\
\operatorname{tr}\{\boldsymbol{AB}\} &= \left(\operatorname{vec}\boldsymbol{A}^{\mathrm{T}}\right)^{\mathrm{T}}\operatorname{vec}\boldsymbol{B}\\
\operatorname{vec}(\boldsymbol{AXB}) &= \left(\boldsymbol{B}^{\mathrm{T}}\otimes\boldsymbol{A}\right)\operatorname{vec}\boldsymbol{X}
\end{aligned}$$

A.3 Projection into the Subspace of Annihilated Column Sums

Proof of lemma 2.4. Using

$$\sum_{i=1}^{n} T_{n;ij} = 0, \qquad \forall j, \tag{A.5}$$

we can verify that

$$\sum_{i=1}^{n} \hat{M}_{ij} = \sum_{i=1}^{n} (\boldsymbol{T}_n \boldsymbol{M})_{ij} = \sum_{i,k=1}^{n} T_{n;ik} M_{kj} = \sum_{k=1}^{n} M_{kj} \underbrace{\sum_{i=1}^{n} T_{n;ik}}_{0} = 0. \tag{A.6}$$

□

A.4 Trivial Fixed Point of the Entropy Barrier Function

Lemma A.1
The minimization problem

$$\begin{aligned} \min_{\boldsymbol{M} \in \mathbb{R}_+^{n \times n}} F(\boldsymbol{M}) &:= \sum_{i,j=1}^{n} M_{ij} \log\left(M_{ij}\right) \\ \textit{subject to} \quad & \sum_{i=1}^{n} M_{ij} = 1, \qquad \forall j \\ & \sum_{j=1}^{n} M_{ij} = 1, \qquad \forall i \end{aligned}$$

has the unique minimum $M_{ij} = \frac{1}{n} \quad \forall i, j$.

Proof. Since F is strictly convex on $\mathbb{R}_+^{n \times n}$ and the constraints are convex too, F has a unique minimum. The corresponding Lagrange function is defined as

$$L(\boldsymbol{M}, \boldsymbol{\mu}, \boldsymbol{\nu}) := \sum_{i,j=1}^{n} M_{ij} \log\left(M_{ij}\right) + \sum_{i=1}^{n} \mu_i \left[\sum_{j=1}^{n} M_{ij} - 1\right] + \sum_{j=1}^{n} \nu_j \left[\sum_{i=1}^{n} M_{ij} - 1\right].$$

When we evaluate the stationary conditions of L, we receive

$$\log\left(M_{ij}\right) + 1 + \mu_i + \nu_j = 0, \qquad \forall i, j.$$

After substituting the above term into the stationary condition of the row sum constraints, we obtain

$$\sum_{i=1}^{n} M_{ij} = \sum_{i=1}^{n} \exp\left(-1 - \mu_i - \nu_j\right) = \exp\left(-1 - \nu_j\right) \sum_{i=1}^{n} \exp\left(-\mu_i\right) = 1, \qquad \forall j.$$

It follows that all ν_j are equal. Making the analog argument for the column sums reveals that the μ_i are equal. The proposition immediately follows.

□

A.5 Bound on Lagrange Parameter Vector

Proof of lemma 2.8. According to the derivation of the soft-assign QAP algorithm, the iterative update scheme from equation (2.38) can be rewritten as

$$M_{ij}^{(r+1)} = u_i v_j W_{ij}^{(r)}, \qquad \forall i, j. \tag{A.7}$$

Using the exact satisfaction of the column sum constraint, we receive that

$$\sum_i M_{ij}^{(r+1)} = \sum_i u_i v_j W_{ij}^{(r)} = 1, \qquad \forall j \tag{A.8}$$

$$\Rightarrow v_j = \frac{1}{\sum_i u_i W_{ij}^{(r)}}, \qquad \forall j. \tag{A.9}$$

Substituting this into equation (A.7) leads to

$$M_{ij}^{(r+1)} = \frac{u_i W_{ij}^{(r)}}{\sum_k u_k W_{kj}^{(r)}} = \frac{1}{1 + \sum_{k \neq i} \frac{u_k W_{kj}^{(r)}}{u_i W_{ij}^{(r)}}} \leq \frac{1}{1 + \min_j \sum_{k \neq i} \frac{u_k W_{kj}^{(r)}}{u_i W_{ij}^{(r)}}}, \qquad \forall i, j. \tag{A.10}$$

From the approximately satisfied row sum constraint $|\sum_j M_{ij}^{(r+1)} - 1| < \epsilon,\ \forall i$, we obtain that

$$1 - \epsilon \leq \sum_j M_{ij}^{(r+1)} \leq \frac{n}{1 + \min_j \sum_{k \neq i} \frac{u_k W_{kj}^{(r)}}{u_i W_{ij}^{(r)}}}, \qquad \forall i. \tag{A.11}$$

Let us rearrange this to

$$\min_j \sum_{k \neq i} \frac{u_k W_{kj}^{(r)}}{u_i W_{ij}^{(r)}} \leq \frac{n - 1 + \epsilon}{1 - \epsilon}, \qquad \forall i, \tag{A.12}$$

which is also valid for each of the summands on the left. Moreover, for $k = i$ it is valid, if $n \geq 2 > 2 \cdot (1 - \epsilon)$, which is true for every reasonable case. Thus,

$$\min_j \frac{u_k W_{kj}^{(r)}}{u_i W_{ij}^{(r)}} \leq \frac{n - 1 + \epsilon}{1 - \epsilon}, \qquad \forall i, k \tag{A.13}$$

$$\Rightarrow \frac{u_k}{u_i} \leq \max_j \frac{W_{ij}^{(r)}}{W_{kj}^{(r)}} \cdot \frac{n - 1 + \epsilon}{1 - \epsilon}, \qquad \forall i, k \tag{A.14}$$

$$\Rightarrow \max_{i,k} \frac{u_k}{u_i} \leq \max_{i,j,k} \frac{W_{ij}^{(r)}}{W_{kj}^{(r)}} \cdot \frac{n - 1 + \epsilon}{1 - \epsilon}. \tag{A.15}$$

From Sinkhorn's theorem 2.1, we know that the scaling coefficients u_i and v_j are unique up to a factor. We exploit this property by requiring that $\Pi_i u_i = 1$. From that, we can conclude that

$$\left.\begin{array}{l} \max_i \ u_i \\ \max_i \ \frac{1}{u_i} \end{array}\right\} \leq \max_{i,j,k} \frac{W_{ij}^{(r)}}{W_{kj}^{(r)}} \cdot \frac{n - 1 + \epsilon}{1 - \epsilon}. \tag{A.16}$$

Taking the logarithm on both sides and multiplying with $\frac{1}{\beta}$ gives

$$\left.\begin{array}{r}\max_i \left(-\mu_i^{(r+1)}\right)\\ \max_i\ \mu_i^{(r+1)}\end{array}\right\} \leq \max_{i,p,k}\left(\sum_{j,l} C^{(\gamma)}_{pj;kl} M^{(r)}_{jl} - \sum_{j,l} C^{(\gamma)}_{ij;kl} M^{(r)}_{jl} + B_{pk} - B_{ik}\right) + \frac{1}{\beta}\log\frac{n-1+\epsilon}{1-\epsilon}. \tag{A.17}$$

If we exchange $\max_{i,p,k}$ and $\sum_{j,l}$, this makes the right side larger. Thus, we obtain that:

$$\max_i |\mu_i^{(r+1)}| \leq \sum_{j,l}\left[\max_{i,p,k}\left(C^{(\gamma)}_{pj;kl} - C^{(\gamma)}_{ij;kl}\right) M^{(r)}_{jl}\right] + \max_{i,p,k}\left(B_{pk} - B_{ik}\right) + \frac{1}{\beta}\log\frac{n-1+\epsilon}{1-\epsilon} \tag{A.18}$$

$$\leq \sum_{l}\left[\max_{i,j,p,k}\left(C^{(\gamma)}_{pj;kl} - C^{(\gamma)}_{ij;kl}\right) \underbrace{\sum_j M^{(r)}_{jl}}_{=1}\right] + \max_{i,p,k}\left(B_{pk} - B_{ik}\right) + \frac{1}{\beta}\log\frac{n-1+\epsilon}{1-\epsilon}. \tag{A.19}$$

Because of exact column sum constraint satisfaction, we finally conclude that

$$\max_i |\mu_i^{(r+1)}| \leq \sum_{l}\max_{i,j,p,k}\left(C^{(\gamma)}_{pj;kl} - C^{(\gamma)}_{ij;kl}\right) + \max_{i,p,k}\left(B_{pk} - B_{ik}\right) + \frac{1}{\beta}\log\frac{n-1+\epsilon}{1-\epsilon}. \tag{A.20}$$

□

A.6 Performance on QAPLIB Instances

Name	*feas. solution*	*w/o 2opt*	*w/ 2opt*	β_0	β_r	β_f	γ	r_r	s_r
Chr12a	9552	14406	9916	0.5	1.15	40.0	0.25	1	1
Chr12b	9742	13888	9742	0.5	1.075	20.0	0	1	2.5
Chr12c	11156	15406	11974	0.5	1.15	40.0	0.25	1	1
Chr15a	9896	12900	10976	0.5	1.075	13.3	0.25	1	1
Chr15b	7990	15776	8640	0.5	1.15	40.0	0.5	1	1
Chr15c	9504	16528	11352	0.5	1.05	20.0	0	1	1
Chr18a	11098	20744	12642	0.5	1.075	20.0	0.25	1	1
Chr18b	1534	1764	1560	0.5	1.15	40.0	0	1	1
Chr20a	2192	3176	2596	0.5	1.075	13.3	0	2.5	1
Chr20b	2298	3628	2670	0.5	1.025	40.0	0	1	1
Chr20c	14142	45126	16762	0.5	1.025	13.3	0.5	2.5	1
Chr22a	6156	8124	6602	0.5	1.15	13.3	0.25	2.5	1
Chr22b	6194	10004	6636	0.5	1.15	40.0	0.25	2.5	1
Chr25a	3796	8256	4574	0.5	1.075	20.0	0.25	1	1

Table A.1: Performance on QAPLIB instances

Name	*feas. solution*	*w/o 2opt*	*w/ 2opt*	β_0	β_r	β_f	γ	r_r	s_r
Els19	17212548	25366272	20265368	0.5	1.15	20.0	0.5	1	2.5
Esc16a	68	72	68	0.5	1.15	20.0	0	2.5	2.5
Esc16b	292	292	292	0.5	1.15	40.0	0	1	1
Esc16c	160	164	160	0.5	1.15	40.0	0	1	2.5
Esc16d	16	18	16	0.5	1.15	40.0	0	1	1
Esc16e	28	32	28	0.5	1.15	40.0	0	2.5	2.5
Esc16g	26	30	26	0.5	1.15	40.0	0	1	2.5
Esc16h	996	996	996	0.5	1.15	40.0	0	1	1
Esc16i	14	14	14	0.5	1.15	40.0	0	1	1
Esc16j	8	14	8	0.5	1.15	40.0	0	1	1
Esc32a	130$*$	152	134	0.5	1.025	20.0	0	1	1
Esc32b	168$*$	192	184	0.5	1.15	13.3	0	2.5	2.5
Esc32c	642$*$	646	642	0.5	1.15	40.0	0	1	1
Esc32d	200$*$	208	200	0.5	1.15	40.0	0	2.5	1
Esc32e	2	2	2	0.5	1.15	40.0	0	1	1
Esc32f	2	2	2	0.5	1.15	40.0	0	1	1
Esc32g	6$*$	6	6	0.5	1.15	40.0	0	1	1
Esc32h	438$*$	446	438	0.5	1.15	13.3	0.75	1	1
Esc64a	116$*$	116	116	0.5	1.15	40.0	0	1	1
Esc128	64$*$	116	70	0.5	1.05	20.0	0.25	1	1
Had12	1652	1778	1660	0.5	1.15	20.0	0	1	1
Had14	2724	2756	2724	0.5	1.05	20.0	0.25	2.5	1
Had16	3720	4028	3720	0.5	1.15	40.0	0.25	1	1
Had18	5358	5702	5368	0.5	1.15	20.0	0.25	1	1
Had20	6922	7258	6934	0.5	1.15	20.0	0.25	1	1
Kra30a	88900$*$	94350	88900	0.5	1.15	13.3	0	2.5	1
Kra30b	91420$*$	107320	92550	0.5	1.15	20.0	0	2.5	2.5
Nug12	578	640	578	0.5	1.15	13.3	0	2.5	1
Nug14	1014	1152	1016	0.5	1.05	13.3	0	1	1
Nug15	1150	1238	1160	0.5	1.075	20.0	0	1	2.5
Nug16a	1610	1800	1638	0.5	1.15	20.0	0.25	2.5	1
Nug16b	1240	1434	1264	0.5	1.15	20.0	0	2.5	2.5
Nug17	1732	1844	1748	0.5	1.075	40.0	0.25	2.5	1
Nug18	1930	2132	1938	0.5	1.025	20.0	0	1	1
Nug20	2570	2848	2596	0.5	1.05	13.3	0	1	2.5
Nug21	2438	2672	2450	0.5	1.025	13.3	0	1	1
Nug22	3596	3778	3596	0.5	1.05	13.3	0.75	1	1
Nug24	3488$*$	3920	3490	0.5	1.025	20.0	0.25	2.5	1
Nug25	3744$*$	4074	3754	0.5	1.05	13.3	0	2.5	2.5
Nug30	6124$*$	6676	6186	0.5	1.05	20.0	0.25	2.5	1
Rou12	235528	257800	238134	0.5	1.15	20.0	0	1	1
Rou15	354210	394248	359748	0.5	1.15	40.0	0	2.5	1

Table A.1: Performance on QAPLIB instances

Name	*feas. solution*	*w/o 2opt*	*w/ 2opt*	β_0	β_r	β_f	γ	r_r	s_r
Rou20	725522	828062	740860	0.5	1.15	40.0	0	1	1
Scr12	31410	38164	31410	0.5	1.15	13.3	0	1	1
Scr15	51140	65632	51140	0.5	1.15	13.3	0	1	2.5
Scr20	110030	129354	111822	0.5	1.05	20.0	0	1	2.5
Sko42	15812∗	18716	16222	0.5	1.05	20.0	0.25	1	1
Sko49	23386∗	26690	23828	0.5	1.05	20.0	0	1	1
Sko56	34458∗	39616	35200	0.5	1.05	20.0	0.25	1	1
Sko64	48498∗	55408	49342	0.5	1.075	20.0	0	1	1
Sko72	66256∗	74212	67222	0.5	1.075	20.0	0	1	1
Sko81	90998∗	100562	92494	0.5	1.075	20.0	0	1	1
Sko90	115534∗	128226	117176	0.5	1.05	20.0	0.25	1	1
Sko100a	152002∗	167464	154302	0.5	1.05	20.0	0.5	1	1
Sko100b	153890∗	170048	155850	0.5	1.05	20.0	0.5	1	1
Sko100c	147862∗	164268	150514	0.5	1.05	20.0	0.5	1	1
Sko100d	149576∗	162574	151788	0.5	1.05	20.0	0.25	1	1
Sko100e	149150∗	164822	151210	0.5	1.05	20.0	0.25	1	1
Sko100f	149036∗	164138	151308	0.5	1.05	20.0	0.25	1	1
Ste36a	9526∗	11140	9804	0.5	1.05	40.0	0.25	1	1
Ste36b	15852∗	19720	16248	0.5	1.025	13.3	0.25	2.5	1
Ste36c	823911?∗	9630080	8393650	0.5	1.025	40.0	0	1	1
Tai12a	224416	245006	230704	0.5	1.025	20.0	0.25	1	1
Tai12b	39464925	50996318	39900385	0.5	1.15	13.3	0.25	1	1
Tai15a	388214	424986	393146	0.5	1.075	13.3	0.5	1	1
Tai15b	51765268	494800943	51969367	0.5	1.15	20.0	0.25	1	1
Tai17a	491812	546228	504468	0.5	1.15	20.0	0	1	1
Tai20a	703482	806858	728078	0.5	1.15	40.0	0.25	1	2.5
Tai20b	122455319∗	140696443	123738722	0.5	1.15	40.0	0	1	1
Tai25a	1167256∗	1360666	1205477	0.5	1.075	40.0	0	1	1
Tai25b	344355646∗	483347219	346705166	0.5	1.15	40.0	0.25	1	1
Tai30a	1818146∗	2052728	1875350	0.5	1.075	40.0	0.5	1	1
Tai30b	637117113∗	834251995	639170392	0.5	1.05	40.0	0	1	1
Tai35a	2422002∗	2787994	2534388	0.5	1.15	40.0	0	1	1
Tai35b	283315445	348651385	285328883	0.5	1.025	40.0	0	1	1
Tai40a	3139370∗	3512334	3256832	0.5	1.15	13.3	0	1	1
Tai40b	637250948∗	827700540	640223865	0.5	1.15	40.0	0	1	2.5
Tai50a	4941410∗	5723066	5172662	0.5	1.05	20.0	0	1	1
Tai50b	458821517∗	567843961	484635218	0.5	1.05	20.0	0.25	1	1
Tai60a	7208572∗	8310636	7498004	0.5	1.05	20.0	0	1	1
Tai60b	608215054∗	821580716	626873523	0.5	1.05	20.0	0.5	1	1
Tai64c	1855928∗	5893540	1864686	0.5	1.05	20.0	0	1	1
Tai80a	13557864∗	15385348	13990144	0.5	1.05	20.0	0.25	1	1
Tai80b	818415043∗	1057927047	840600327	0.5	1.075	20.0	0	1	1

Table A.1: Performance on QAPLIB instances

Name	*feas. solution*	*w/o 2opt*	*w/ 2opt*	β_0	β_r	β_f	γ	r_r	s_r
Tai100a	21125314$*$	23782192	21755244	0.5	1.05	20.0	0.5	1	1
Tai100b	1185996137$*$	1420942964	1212014719	0.5	1.05	20.0	0.5	1	1
Tai150b	499348972$*$	609689786	514934358	0.5	1.05	20.0	0.5	1	1
Tho30	149936$*$	161054	151896	0.5	1.025	13.3	0.5	1	1
Tho40	240516$*$	256284	243416	0.5	1.075	40.0	0.25	1	2.5
Tho150	8134030$*$	9148466	8275938	0.5	1.05	20.0	0.5	1	1
Wil50	48816$*$	53164	49498	0.5	1.05	20.0	0.25	1	1
Wil100	273038$*$	290534	274458	0.5	1.05	20.0	0.5	1	1

Table A.1: Performance on QAPLIB instances

Appendix B
Bibliography

[1] FEX. URL `http://www.ipb.uni-bonn.de/ipb/projects/fex/fex.html`.

[2] Rainer E. Burkard, Eranda Çela, Panos M. Pardalos, and Leonidas S. Pitsulis. The quadratic assignment problem. In D.-Z. Zhu and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 3, pages 241–337. Kluwer, 1998.

[3] Rainer E. Burkhard, Stefan E. Karisch, and Franz Rendl. QAPLIB-A quadratic assignment problem library. *European Journal of Operational Research*, 55:115–119, 1991. URL `http://www.imm.dtu.dk/~sk/qaplib/`.

[4] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley and Sons, 1998.

[5] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[6] Steven Gold, Eric Mjolsness, and Anand Rangarajan. Clustering with a domain-specific distance measure. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 96–103. Morgan Kaufman Publishers, 1994.

[7] Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, April 1996.

[8] Steven Gold and Anand Rangarajan. Softmax to softassign: Neural network algorithms for combinatorial optimization. *Journal of Artificial Neural Networks*, 2(4): 381–399, August 1996.

[9] Alexander Graham. *Kronecker Products and Matrix Calculus With Applications*. Mathematics and its Applications. Ellis Horwood, 1981.

[10] Shin Ishii and Hirotaka Niitsuma. λ-opt neural approaches to quadratic assignment problems. *Neural Computation*, 12(9):2209–2225, 2000.

[11] Shin Ishii and Masa-aki Sato. Constrained neural approaches to quadratic assignment problems. *Neural Networks*, 11:1073–1082, 1998.

[12] Shin Ishii and Masa-aki Sato. Doubly constrained network for combinatorial optimization. *Neurocomputing*, 2001.

[13] Pascal Koiran. Dynamics of discrete time, continuous state Hopfield networks. *Neural Computation*, 6(3):459–468, 1994.

[14] Tjalling C. Koopmans and Martin Beckman. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.

[15] J.J. Kosowsky and A[lan] L. Yuille. The invisible hand algorithm: Solving the assignment problem with statistical physics. *Neural Networks*, 7(3):477–490, 1994.

[16] E. L. Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963.

[17] S. Z. Li. Matching: Invariant to translations, rotations and scale changes. In *Pattern Recognition*, volume 25, pages 583–594, June 1992.

[18] David G. Luenberger. *Linear and nonlinear programming*. Addison-Wesley, 2nd edition, 1984.

[19] Eric Mjolsness and Charles Garrett. Algebraic transformations of objective functions. *Neural Networks*, 3:651–669, 1990.

[20] Katta G. Murthy. *Linear Complementarity, Linear and Nonlinear Programming*. Sigma Series in Applied Mathematics. Heldermann Verlag, 1988.

[21] Panos M. Pardalos, Franz Rendl, and Henry Wolkowicz. The quadratic assignment problem: A survey and recent developments. In Panos M. Pardalos and Henry Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–42. AMS, 1994.

[22] Carsten Peterson and Bo Söderberg. A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, 1(1):3–22, 1989.

[23] Carsten Peterson and Bo Söderberg. Artificial neural networks. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 7, pages 173–213. John Wiley and Sons, 1997.

[24] Anand Rangarajan, Steven Gold, and Eric Mjolsness. A novel optimizing network architecture with applications. *Neural Computation*, 8(5):1041–1060, 1996.

[25] Anand Rangarajan, Alan Yuille, Steven Gold, and Eric Mjolsness. A convergence proof for the softassign quadratic assignment algorithm. *Advances in Neural Information Processing Systems*, 9:620–626, 1997.

[26] Anand Rangarajan, Alan Yuille, and Eric Mjolsness. Convergence properties of the softassign quadratic assignment algorithm. *Neural Computation*, 11(6):1455–1474, 1999.

[27] Masa-aki Sato and Shin Ishii. Bifurcations in mean-field-theory annealing. *Phycial Review E*, 53:5153–5168, 1996.

[28] Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *Annals of Mathematical Statistics*, 35(8):876–879, 1964.

[29] Steven Skiena. *Implementing Discrete Mathematics. Combinatorics and Graph Theory with Mathematica.* Addison-Wesley, 1990.

[30] Todd L. Veldhuizen. Arrays in Blitz++. In *Proceedings of the 2nd International Scientific Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, Lecture Notes in Computer Science. Springer Verlag, 1998.

[31] T[odd] L. Veldhuizen and M. E. Jernigan. Will C++ be faster than Fortran? In *Proceedings of the 1st International Scientific Computing in Object-Oriented Parallel Environments (ISCOPE'97)*, Lecture Notes in Computer Science. Springer Verlag, 1997.

[32] A[lan] L. Yuille and J. J. Kosowsky. Statistical physics algorithms that converge. *Neural Computation*, 6(3):341–356, 1994.

Declaration

I hereby certify that this diploma thesis was written without the help of a third person. I further assure that all sources and tools used are clearly indicated in this work.

This thesis has not been submitted in this or in any other form to any authority of examination.

Mannheim, Mai 17, 2001