TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

D I P L O M A R B E I T

Eine generische Bibliothek für Metaheuristiken und ihre Anwendung auf das Quadratic Assignment Problem

ausgeführt am

Institut für Computergraphik und Algorithmen 186

der

Technischen Universität Wien

unter Anleitung von

a.o. Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl

durch

Daniel Wagner

Schauleithenstraße 9

3363 Ulmerfeld-Hausmening

Datum

Unterschrift

Abstract

In this master thesis a generic libray of efficient metaheuristics for combinatorial optimization is presented. In the version at hand classes that feature local search, simulated annealing, tabu search, guided local search and greedy randomized adaptive search procedure were implemeted.

Most notably a generic implementation features the advantage that the problem dependent classes and methods only need to be realized once without targeting a specific algorithm because these parts of the sourcecode are shared among all present algorithms contained in EAlib.

This main advantage is then exemplarily demonstrated with the quadratic assignment problem. The sourcecode of the QAP example can also be used as an commented reference for future problems.

Concluding the experimental results of the individual metaheuristics reached with the presented implementation are presented.

Kurzfassung

In dieser Diplomarbeit wird eine generische Bibliothek von effizienten Metaheuristiken für kombinatorische Optimierungsprobleme vorgestellt. In der vorliegenden Version enthält sind lokale Suche, Simulated Annealing, Tabusearch, Guided Local Search und Greedy Randomized Adaptive Search Procedure implementiert worden.

Eine generische Implementierung bietet vorallem den Vorteil das bei einem neuen zu lösendem Problem nur einige bestimmte problemabhängige Klassen und Methoden realisiert werden müssen ohne sich schon im Vorhinein einen speziellen Algorithmus festzulegen, da diese Klassen und Methoden von allen in der EAlib vorhanden Metaheuristiken verwendet werden.

Die Vorteile dieser Bibliothek werden anschließend anhand des Quadratic Assignment Problems ausführlich dargestellt. Dieses Beispiel dient zusätzlich auch noch als kommentierte Referenz für zukünftige Problemimplentierungen.

Abschließend werden die Resulate der Experimente mit den verschiedenen Metaheuristiken präsentiert.

Danksagung

An dieser stelle möchte ich mich bei allen Menschen bedanken die zum Gelingen dieser Diplomarbeit beigetragen haben.

Dieser Dank gilt meinem Betreuer Prof. Raidl, der mich mit großer Geduld am Weg zum Abschluß begleitet hat und mit mir in den vielen Treffen oft nützliche Ideen entwickelt hat.

Meinen Eltern und meinem Bruder Ronald danke ich für ein sorgloses Studium und die moralische Unterstützung wenn die Motivation einmal nicht so groß war.

Bei meinen Studienkollegen, besonders bei Harry und Zamb, bedanke ich mich für die Freundschaft, den Spaß und die gegenseitige Unterstützung.

Last but not least möchte ich mich auch bei meinen Mitbewohnern Sic0 und Leo bedanken, die mir während meiner Arbeit die nötige Ruhe zukommen ließen, aber natürlich auch ab und zu für willkommene Ablenkung gesorgt haben.

Natascha danke ich für die schöne gemeinsame Zeit.

Table of Contents

All men by nature desire knowledge.

Aristotle

Chapter 1

Introduction

1.1 Motivation

Metaheuristics are a popular approach to handle computationally intractable optimization problems. In the course of this master thesis an existing library dedicated to evolutionary algorithms was extended substantially by several common known and used metaheuristics. These metaheuristics are implemented in a generic manner so that their application to a widespread variety of combinatorial optimization problems is supported.

A generic implementation of metaheuristics is desirable because common portions of many metaheuristics can be implemented problem independent and also a significant amount of problem dependent sourcecode can be shared between the metaheuristics, e.g. efficient evaluation of the objective value or neighborhood relevant methods.

The basis for the implementation of the metaheuristics is the EAlib library which is developed at the Vienna University of Technology, Institute of Computergraphics and Algorithms. At the beginning of this master thesis it already contained particular classes for evolutionary algorithms and some supporting infrastructure which was also useful for our project. The aim of this master thesis the was to extend this existing library while trying to keep changes to the existing parts to a minimum to maintain compatibility with present applications.

1.2 Combinatorial Optimization and Metaheuristics

An optimization problem can be characterized as the selection of a "best" configuration or set of parameters to achieve some objective criteria. If the entities to

be optimized are discrete, the number of feasible solutions is finite. We call such problems combinatorial optimization problems.

A combinatorial optimization problem is specified formally by a set of problem instances and is either a minimization problem or a maximization problem. An instance of a combinatorial minimization problem is a pair $(\mathcal{X}, f)$, where the solution set $\mathcal{X}$ is the set of all feasible solutions and the cost function f is a mapping $f : \mathcal{X} \leftarrow R$. The problem is to find a globally optimal solution, i.e. an $x^* \in \mathcal{X}$ such that $f(x^*) \leq f(x)$ for all $x \in \mathcal{X}$. Maximization problems can be trivially transformed into minimization problems by changing the sign of the cost function f.

Salient examples are the traveling sales problem and related routing and transportation problems, scheduling and time-tabling, cutting and packing tasks. Most of these problems are NP-hard. However NP-hardness does not necessarily mean that all practically relevant instances are not solveable within acceptable time. Vice versa, an algorithm for a polynomial-time solvable problem might be too expensive in practice.

Many different algorithmic strategies exist to deal with this problems and the metaheuristics, which are the main topic of this work, are among of them. Traditionally metaheuristics are considered as solution methods utilizing an interaction between local improvement procedures and higher level strategies to overcome local optima leading to a robust search process. In general metaheuristics contain are not designed for a specific optimization problem. They rather can be applied to a wide range of problems. Therefore many metaheuristics can be implemented in a generic manner straighforward.

For the library at hand five initial metaheuristics were chosen for implementation which are local search, simulated annealing, tabu search, guided local search and greedy randomized adaptive search procedures.

1.3 Guide to the thesis

The thesis at hand describes the quadratic assignment problem in Chapter 2 which we chose as an example problem to demonstrate the application of EAlib to a new task and to illustrate the pros and cons of the implemented metaheuristics. In Chapter 3 all featured algorithms are explained. The requirements of functionality, design and usability of the targeted library are specified in Chapter 4 while the details of the implemented library are stated in Chapter 5. Finaly experimental results of solving the quadratic assignment problem using the new EAlib are presented in Chapter 6.

Science is organized knowledge. Wisdom is organized life.

Imanuel Kant

Chapter 2

Quadratic Assignment Problem

Since the quadratic assignment problem (QAP) was mentioned first by Koopmans and Beckmann [23] in 1957, they used the QAP to model economic activities, many authors contributed to it, see Loiola et al. [27] for a recent survey article about the QAP. The major attraction points of the QAP are its practical and theoretical importance and its computational complexity — it is one of the most difficult combinatorial optimization problems. In general problem instances of size $n \geq 30$ can not be solved in reasonable time. Sahni and Gonzales [39] had first shown that the QAP is a member of the class of NP-hard problems and that, unless P = NP, it is not possible to find a polynomial ϵ-approximation algorithm, for a constant ϵ. Nevertheless recent results (Gutin and Yeo [20]) proved that, in the case of QAP, polynomial approximations with factorial domination number exist. For more information on the theory of NP-completeness Garey and Johnson [14] is recommended.

Since the QAP is very versatile, several other NP-hard combinatorial optimization problems such as traveling salesman problem (TSP), graph partitioning, the bin-packing problem (BPP) or the max clique problem can be formulated and solved using QAPs [5, 27].

Prior to an exact definition of the QAP, a simpler related problem, the linear assignment problem (LAP), is presented as a smoother introduction assignment. After a short description of the LAP, a comprehensive explanation of the QAP, which will cover a problem definition and various mathematical formulation approaches, resolution methods and finally applications, will be provided.

2.1 Problem Description

Assigning objects is a common task for econimic or techinical staff. Therefore it is not a surprise that assigment problems are among the greatest challanges in the area of combinatorial optimization.

As an introduction the linear assignment problem (LAP) is presented here. Assume there are two equal sized sets of objects, e.g. persons and jobs, and they are assigned to each other by making up pair of those objects, taking one from each set for a pair. Additionally every possible pair is given a value, which results in a $n \times n$ matrix with n^2 elements. The problem now is to find an assignment of all objects for which the sum of the values is minimized. An example application for the LAP is the assignment of persons to jobs.

Mathematically this problem can be formulated as follows.

$$\min_{\pi \in \Pi} \sum_{i=1}^{n} a_{i,\pi(i)} \tag{2.1}$$

where $A = [a_{i,\pi(i)}]$ is the matrix of values for assigning object i to $\pi(i)$ and further Π is the set of all permutations of the n elements $\{1, \ldots, n\}$. The LAP is polynomial and is easily solved by the Hungarian method [27] which was proposed by Harold W. Kuhn in 1955 [24].

Reconsidering the above description the question arises if it really true that an assignment of two objects does not have any sideeffects on other assignments. If this assumption does not hold, the quadratic assignemnt problem may give an appropriate formal description of the real-world problem.

QAP is a generalization of in the linear assignment problem in a manner that assignment can affect each another. Therefore, in addition to the value matrix — when using QAPs it is called distance matrix — a flow matrix of same dimension is introduced. As an example that is related to the previous mentioned one with persons and jobs, the distance matrix can be interpreted as the distance between the offices and the flow as the amount of interaction between these persons.

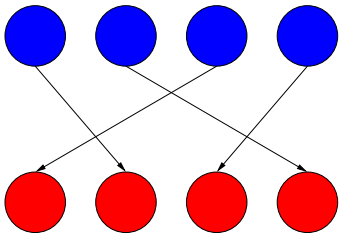


Figure 2.1: A quadratic assignment example

2.2 Formulations

Nowadays many different formulations are used. Loiola et al. [27] and Commander and Pardalos [9] give a good survey over the existing formulations of the quadratic

assignment problem, different resolution methods, lower bound calculation and applications.

2.2.1 Permutation Formulation

As an introduction the popular and very intuitive formulation is based on permutations is given. Thereby the QAP can be stated as follows. Let A, B and C be $n \times n$ matrices representing flows between objects, distances between locations and costs for assigning objects to locations, further let Π be the set of all possible permutations of the n elements $\{1, \ldots, n\}$.

$$\min_{\pi \in \Pi} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}\, b_{\pi(i),\pi(j)} + \sum_{i=1}^{n} c_{i,\pi(i)} \tag{2.2}$$

$a_{i,j}$ is the flow between objects i and j, $b_{\pi(i),\pi(j)}$ is the distance between locations $\pi(i)$ and $\pi(j)$ and $c_{i,\pi(i)}$ is the fixed cost of assigning object i to location $\pi(i)$.

The formulation given contains a linear part to model fixed assignment cost. However many authors neglect this term of the equation, since it is a LAP and thus easy to be solved, e.g. with the Hungarian method, or because they do not need this term for their considerations; the resulting formulation is stated below:

$$\min_{\pi \in \Pi} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j} b_{\pi(i),\pi(j)} \tag{2.3}$$

In the implementation of this master thesis we used the term to be minimized in the above formula as objective function. Consequently our solution representation consists of the permutation vector π.

2.2.2 Integer Linear Programming

Koopmans and Beckman [23] used a different formulation in their initial statement of the quadratic assignment problem; the so-called integer linear programming (IP) formulation. It is still of great use, since IP is a topic of ongoing research. In this formulation the reader also can see why the problem is called quadratic, which is not so obsious in some of the other formulations.

The general IP formulation is as follows. Let $A = [a_{i,j}]$ be a matrix of flows between objects i and j and further $B = [b_{k,p}]$ a matrix of the distances between positions k and p and lastly $C = [c_{i,k}]$ a matrix of costs for assigning object i to

position k:

$$\min \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} a_{i,j} b_{k,p} x_{i,j} x_{k,p} + \sum_{i,k=1}^{n} c_{i,k} x_{i,k} \tag{2.4}$$

$$\text{s.t.} \sum_{i=1}^{n} x_{i,j} = 1 \qquad 1 \leq j \leq n, \tag{2.5}$$

$$\sum_{j=1}^{n} x_{i,j} = 1 \qquad 1 \leq i \leq n, \tag{2.6}$$

$$x_{i,j} \in \{0,1\} \qquad 1 \leq i,j \leq n. \tag{2.7}$$

The actual QAP is the problem of minizing equation above, by proper choice of the permutation matrix $X = [x_{i,j}]$. The minimand contains a term of second degree in the unknown permutation matrix X and therefor the problem is called quadratic.

For the same reason as in the prior section the linear term regarding the fixed costs of assigning objects to locatinos can be neglected, leading to the following formulation:

$$\min \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} a_{i,j}\, b_{k,p}\, x_{i,j}\, x_{k,p} \tag{2.8}$$

$$\text{s.t. } (2.5), (2.6) \textit{ and } (2.7).$$

2.2.3 Trace Formulation

Since the essential information about an actual QAP instance is represented usually with matrices it is not surprising that a formulation evolved which takes advantage of this; the trace formulation is an approach to mathematically describe the QAP that uses the trace of a matrix which is defined by trace $A = \sum_i^n a_{i,i}$. It was introduced by Edwards [10]. Again consider $A = [a_{i,j}]$ a matrix of flows from object i to object j, $B = [b_{k,p}]$ distances of location k and p and $C = [c_{i,k}]$ costs of assigning object i to location k.

$$\min_{X \in \Pi} \text{trace}\,(AXB^T + C)X^T \tag{2.9}$$

repectively with the linear term of the problem omitted:

$$\min_{X \in \Pi} \text{trace}\,(AXB^T)X^T \tag{2.10}$$

where Π is the set of all $n \times n$ permutation matrices. It is often used in lower bounds related publications.

2.3 Lower Bounds

The knowledge of lower bounds is fundamental when developing optimization algorithms to solve combinatorial or other mathematical problems. This importance of lower bounds is two-fold. At first they are an essential part of exact algorithms, e.g. branch-and-bound procedures. These methods, while attempting to guarantee the global optimum, also try to avoid the total enumeration of the complete search space. Therefore the performance of such methods depends strongly on the computational quality and efficiency of the utilized lower bounding techniques. An other application of lower bounds is the evaluation of the quality of solutions obtained by some heuristic algorithms (see Section 6.1 on page 52).

The quality of the lower bound can be measured by the gap between the computed bound with the known optimal solution, this referred to as the *tightness* of the bound, i.e. good lower bounds are closer to the global optimum. For an exact algorithm a good bounding technique, which can find the bounds quickly[1], should be used. When used in heuristics, lower bound quality is the most important property.

One of the first suggested and best known lower bounds for the quadratic assignment problem is the one presented by Gilmore [15] and Lawler [25]. The Gilmore-Lawler-Bound (GLB) is given by the solution of linear assignment problem whose cost matrix is gained by special inner products of the flow- and distance-matrix of the original QAP. The advantage of the GLB is that is simple and it can be computed efficiently. However, its drawback is that the gap to the optimal solution grows with the size of problem. For this reason the GLB is a weak bound for larger problem instances.

Due to an intensive research activity many other lower bounds have been discoverd. Bounds based on mixed integer linear programming (MILP) relaxations, eigenvalues of the flow- and distance matrix, reformulations of the above mentioned GLB exist. Some of them, e.g. eigenvalue based bounds, really outperform the original GLB so far tightness is concerned, but they suffer from high computation requirements. The most recent and promising research trends are based on semidefinite programming (SDP), reformulation linearization. Anstreicher and Brixius [1] presented a lower bound for the QAP based on semidefinite and convex quadratic programming, a bound using the bundle method is proposed by Rendl and Sotirov [36].

[1] Up to now no bound that features both advantages, tightness and computational cheapness has been discovered.

2.4 Solution Methods

Since its statement, many different approaches were applied to solve the quadratic assignment problem. These can be categorized in either exact or heuristic methods. In this section we an overview about some of the most successfull or frequent used methods of these categories are presented.

2.4.1 Exact Algorithms

The oldest and simplest way, to resolve the quadratic assignment problem, is *enumeration*. This causes evaluation of the objective function for all $n!$ possible permutations and memorizing the best found solutions; note that there is not necesssarily only oneoptimal solution. The computational effort for evaluating the cost of a permutation requires $O(n^2)$ steps, which has to be computed $O(n!)$ times yielding exponentially sized computation times. Enumeration is very simple to code and has small memory requirements, on the other hand its use is very limited and not of practical relevance.

Other methods include quadratic programming, which reformulates the problem as a 0–1 program (see Section 2.2.2 on page 9) and linear programming, which linearizes the QAP by introducing new variables, the resulting linear program can be solved e.g. with mixed integer linear programming methods.

Many of the above methods share the same problem; they vastly examine the complete search space and therefore, as mentioned, only small problem instances can be solved within a reasonable amount time.

The most successful exact resolution methods for the quadratic assignment problem incorporate branch-and-bound (BB) algorithms. Essential for BB is a good bounding technique, because this directly affects the extent to which the search space must be enumerated; the thighter the used bound, the more solutions can be excluded from the exploration.

Branch-and-bound methods attract many researchers due to their potential. For example Frazer [13] and Brixius and Anstreicher [5] describe a BB implementation and Anstreicher et al. [2] describe a grid enabled BB implementation which was used to solve a problem instance of size 30 to optimality. They report the utilization of an average of 650 worker machines over a one-weekend period, which provides the equivalent of almost 7 years of computation on one single HP9000 C3000 workstation. For an other instance of the same size they utilized the equivalent of 15 years on a single C3000. These examples show the potential of parallelization, which is currently one of the major fields of interest.

2.4.2 Heuristics

Heuristic algorithms, contrary to exact algorithms, can not provide any guarantee of optimality for the best solution obtained. The reason for the current research on suboptimal solution methods is the fact that many of them can provide good solutions within reasonable time constraints, which is often necessary real-world application environments. Heuristic methods include the following categories: *constructive*, *enumeration* and *improvement* methods.

Constructive methods, which are among the earliest heuristics to solve the QAP, try to complete a permutation with each iteration of the algorithm. The selection of each assignment is based on a heuristic selection criterion. For example Gilmore [15] introduced one of the first constructive algorithms. Nowadays this category of heuristics focuses new interest because metaheuristics, such as the greedy randomized adaptive search procedure (see Section 3.5 on page 27) incorporate them.

Enumerative methods are motivated by the expectation that an acceptable solution can be found early during a brute force exploration of the search space. For interesting problems these methods do not enumerate the all feasible solutions and therefore different termination criteria are used. Usually the number of total iterations, or iterations between successive improvements is used, other common criteria include a limit on the total execution time or lowering the upper bound when no further improvements are possible after a number of iterations. It is important to remind that any of these termination criteria can prohibit the finding of an optimal solution.

Improvement methods correspond to local search algorithms (see Section 3.1 on page 19. Most of the heuristics for the QAP are part of this category.

An other worthy to mention category of methods are *approximate algorithms*, which are heuristics provinding quality guarantees for their solutions.

2.4.3 Metaheuristics

Metaheuristics are, as their name suggests, heuristic algorithms too, but usually they can be adapted straighforward to a wide range of different problems; this is in general not possible for traditional heuristics. However, as the main focus of this master thesis lays on metaheuristcs we address them extensively in the next chapter.

2.4.4 Research Trends

Current state of the art algorithms can be divided into two major categories, at one side the search for optimal solutions and exact algorithms which can provide them, and on the other side methods that can provide solutions that are good enough in reasonable time. Of course also theoretical developments are of interest.

The main research focus for the QAP is generated by the growing interest on metaheuristics since the end of the 1980's because it is a popular benchmark to compare algorithms. With recent generations of computer technology the QAP attracted new attention, which lead to honorable developments in parallel algorithms.

Promising future developments seem to be possible through the hybridization of several algorithms, which generated some interest in the past, together with parallelization.

2.5 Applications

The initial motivation that lead to the formulation of the quadratic assignment problem was:

> In the light of the practical and theoretical importance of indivisibilities, it may seem surprising that we possess so little in the way of successful formal analysis of production problems involving indivisible resources. (Koopmans and Beckmann [23])
>
> [...]
>
> The assumption that the benefit from an economic activity at some location does not depend on the uses of other locations is quite inadequate to the complexities of locational decisions.

As the quoted statement suggests, a main field applications is allocation of resources with complex interactions of the individual resources. Koopmans and Beckmann were economists and therefore their focus was on economic activities. Example applications are scheduling of jobs or production lines, facility organization, hospital layout. Nevertheless the QAP is also of practical use where it is not so obvious like dartboard design or typewriter layout. Not to forget many engineering applications. In the remainder of this section we illustrate two applications of the quadratic assignment problem in detail.

2.5.1 Steinberg Wiring Problem

In a 1961 paper [40], Leon Steinberg proposed a *backboard wiring* problem. The problem is about the optimal placement of computer components on a backboard in

such a manner, that the total interconnecting wiring length is minimized. Improved wiring length has two main advantages, most important it increases the performance of the designed system, not less attractive are the decreased manufacturing costs. The original problem instance consisted of 34 components with a total of 2625 interconnections which were to be placed on a backboard with 36 open positions (circles in Figure 2.2).

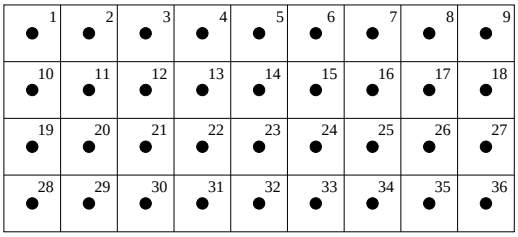


Figure 2.2: Original Backboard of the Steinberg Wiring Problem

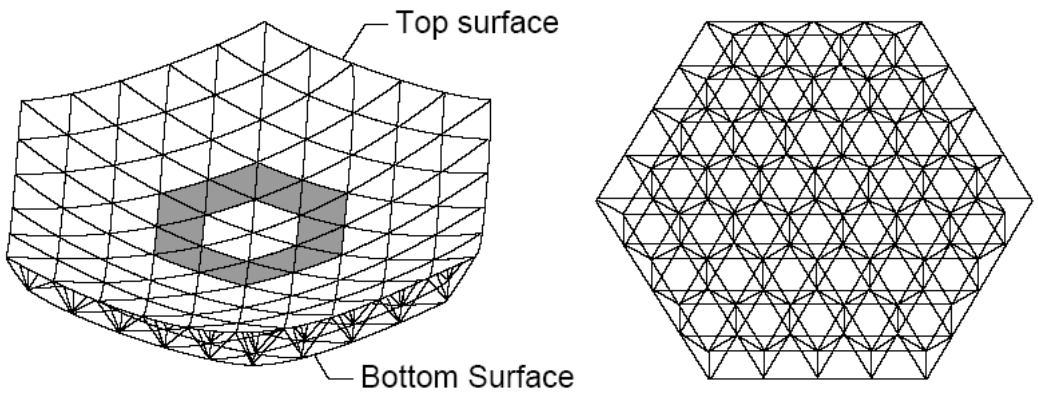
Two *dummy* components, with no connections to any other components, are added so that the number of components equals the number of open positions. The use of dummy elements is a common trick to be able to formulate real-world problems as QAPs. With this addition the mathematical formulation can be given

$$\begin{aligned} \min & \sum_{i,j,k.l} a_{i,k}\, b_{j,l}\, x_{i,j}\, x_{kl} && && (2.11)\\ \text{s.t.} & \sum_{j} x_{i,j} = 1 && i = 1, \ldots, n \\ & \sum_{i} x_{i,j} = 1 && j = 1, \ldots, n \\ & x_{i,j} \in 0, 1 && i, j = 1, \ldots, n \end{aligned}$$

where $a_{i,k}$ is the number of wires interconnection components i and k, $b_{j,l}$ is the distance between positions j and l on the backboard and $x_{i,j} = 1$ if component i is placed at position j. Special attention is payed on the choice of the $b_{j,l}$. In the original paper Steinberg considered using 1-norm, 2-norm or squared 2-norm distances. He further concentrated on obtaining good solutions for the 2-norm and squared 2-norm versions of the problem. However, research interest has been directed to the 1-norm version, which was also used by Brixius and Anstreicher [6] who solved the initial problem instance to optimality with an exact branch-and-bound algorithm, 40 years after its statement. The solution required approximately 186 hours of CPU time on a single Pentium III personal computer.

2.5.2 Antenna Assembly Sequence Problem

At the National Aeronautics and Space Administration (NASA) another interesting application of the quadratic assignment problem is reported by Padula and Kincaid [33]. It is known that NASA often has to design and erect antennas (see Figure 2.3(a)) in space for different purposes like communication with spacecrafts (Deep Space Network). Such an antenna consists of a very large number n of truss elements. For research purposes, the antenna structure is designed as a tetrahedral truss with a flat top surface, which means that all nodes in the top surface of the finite-element model are coplanar (see Figure 2.3(b)). To minimize surface distortions and to the avoid internal forces during the assembly process of the antenna, the truss elements have to be of identical length. However, due to limitations in the manufacturing process, the length is never precisely identical. Each truss element j has a small but measurable error e_j. To overcome the impact of these errors, the truss elements are assembled in such a way, that the errors offset each other.



(a) Antenna configuration (b) Finite element model

Figure 2.3: Conceptual design of a large space antenna (from [33])

For a mathematical formulation of the described problem of arranging the truss elements first, an objective value has to be defined. The objective value of a concrete arrangement is stated as the squared L_2 norm of the surface distortion:

$$\begin{aligned} d^2 &= e^T U^T D U e \\ &= e^T H e \end{aligned} \tag{2.12}$$

where e is the vector of measured errors, U is the influence matrix such that $u_{i,j}$ gives the influence of a truss length error in element j on the surface at node i and D is a positive semidefinite weighting matrix that denotes the relative importance of each node i at which distortion is measured. The calculation of matrix U is can

be done with any structural analysis software package and the matrix D is often the identity matrix. Summarizing this, the combinatorial optimization problem for minimizing antenna distortions is stated as:

$$\min_{e \in E} \sum_{j=1}^{n} \sum_{i=1}^{n} e_i \, h_{i,j} \, e_j \tag{2.13}$$

where E are all possible permutations of the error vector e. Clearly the formulation above is a quadratic assignment problem, although it is not a common formulation; compare the permutation formulation in equation 2.3 on page 9.

In case of the antenna assembly sequence problem simulated annealing and tabu search where applied successfully to solve the problem. Prior to this attempts a pairwise interchange heuristic was suggested, which was based on a simple basic local search algorithm. It is not very surprising that the results achieved with local search where inferior to the ones obtained by simulated annealing or tabu search.

The main advantage for NASA gained by metaheuristically optimized assembling of the truss elements standard precision is adequate which decreases the overall costs since cost for truss elements increase dramatically when unusual precision in length is required.

This example shows that an engineering description of a problem can lead directly to a convenient solution method; however this is not the usual case.

For a successful technology, reality must take precedence over public relations, for Nature cannot be fooled.
Richard Feynman

Chapter 3

Metaheuristics

During the last decades a new kind of heuristic algorithms has emerged which tries to use lower-level heuristic approaches to build higher-level frameworks targeted at efficiently and effectively exploring a search space. The name metaheuristic, first introduced in Glover [16], stems from the composition of two Greek words. *Heuristic* derives from the verb *heuriskein* ($\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$) which means "to find" and the prefix *meta* means "beyond, in an upper level".

This category of algorithms includes[1] Evolutionary Computing (EC) and Genetic Algorithms (GA), Guided Local Search (GLS), Greedy Randomized Adaptive Search Procedure (GRASP), Iterated Local Search (ILS), Simulated Annealing (SA), Tabu Search (TS), Variable Neighborhood Search (VNS) and many more.

For example, Glover and Kochenberger [19] and Blum and Roli [4] provide a survey on metaheuristics and related topics and current state of the art in the area. In this chapter we focus on the concepts and fundamental principles of the metaheuristics implemented during this master thesis.

But before we start off some some terms need to be clearyfied. We consider a *neighborhood structure* as a function $\mathcal{N} : \mathcal{X} \rightarrow 2^{\mathcal{X}}$, which assigns each valid solution $x \in \mathcal{X}$ a set of *neighbors* $\mathcal{N}(x) \subseteq \mathcal{X}$. The set $\mathcal{N}(x)$ is commonly named the *neighborhood* of x. It is usually defined implicity through valid changes (*moves*) on the solutions $x \in \mathcal{X}$.

Furthermore we introduce a *search space*, i.e. a solution representation and an objective function. In other words a search space is a collection of possible solutions to the problem at hand, incorporation some notion of distance between the candidate solutions.

[1] In alphabetical order.

3.1 Basic Local Search

Basic *local search* (LS) is also called *iterative improvement* or *hill-climbing* because at each iteration a move is only performed when the new solution is better than the current solution, regarding to a defined objective function. A move is defined as the selection of a solution s' out of a neighborhood $\mathcal{N}(s)$ of a solution s.

procedure BASIC LOCAL SEARCH
  $s \leftarrow$ GenerateInitialSolution()
  repeat
    $s' \leftarrow$ ChooseNeighbor($\mathcal{N}(s)$)
    if $f(s') \leq f(s)$ **then**
      $s \leftarrow s'$
    end if
  until termination conditions met
end procedure

Algorithm 1: Basic Local Search

In Algorithm 1 the basic algorithm is outlined in pseudocode. First of all the most important task is to define a search space. This means a representation of real-world objects and an objective function f are needed. Regarding the chosen representation an appropriate neighborhood structure has to be found. A popular choice for many combinatorial optimization problems is the *2-opt*[2] neighborhood because it can be applied easy to many problems. Nevertheless, despite some exemptions, 2-opt tends to get stuck in local optima. Some other neighborhoods are *k-flip* for binary strings where the neigborhood consits of all solutions that have a Hamming-Distance less or equal to k. A generalized 2-opt, *k-opt* is also known.

The *GenerateInitialSolution* function is needed to generate an initial solution at which the search begins. This could happen simply by a completely random choice or a more sophisticated construction method. As *ChooseNeighbor($\mathcal{N}(s)$)* function, also called *step function*, theoretically any function that chooses a solution s' out of a neighborhood $\mathcal{N}(s)$ of solution s is possible, but it has turned out that only a few are commonly used:

random neighbor picks a neighboring solution out of $\mathcal{N}(s)$ at random.

first improvement systematically searches $\mathcal{N}(s)$ and chooses the first neighboring solution that is better than s.

best improvement completely explores $\mathcal{N}(s)$ and takes the best neighboring solution.

[2] A 2-opt move consists of removing two edges of a given solution and reconnect them in a different way.

Finally the termination conditions have to be defined. In case of the latter two *ChooseNeighbor($\mathcal{N}(s)$)* functions the simple condition *stop if no further improvement is made* will almost always only find a local optimum. Other possible termination conditions depend on the amount of passed CPU-Time, number of iterations, number of iterations since the last improvement or any combination of these or other conditions, which is virtually always desired.

Depending on the chosen neighborhood the basic local search algorithm often only yields poor locally optimal solutions and is therefore only of limited use. To address this weakness, many advanced local search methods where proposed. Among others iterated local search [28, 29], multi-start methods [30], guided local search, greedy randomized adaptive search procedure, simulated annealing and tabu search have been developed.

3.2 Simulated Annealing

Simulated annealing (SA) was the first major attempt to improve basic local search, which does not perform well if caught in a local optima — as pointed out in in the last section. It was proposed independently by Kirkpatrick et al. [22] and Cerny [8] during the early 1980s and it is commonly said that SA is the oldest among the metaheuristics. Simulated annealing is inspired by the physical process of cooling crystalline matter, hence it is often referred to as a nature inspired method.

The fundamental idea of simulated annealing is that in contrary to basic local search moves resulting in solutions of worse quality than the current solution are allowed with a certain probability in order to escape from local optima; these moves are referred to as uphill moves. The probability of accepting an uphill move depends on the actual deterioration and the current temperature, which is decreased during the search process. The simulated annealing metaheuristic is outlined as pseudocode in Algorithm 2 on the following page.

At first the algorithm generates an initial solution either randomly or with some construction heuristic and initializes the so-called *temperature parameter* T and the counter t. Then at each iteration of the annealing process a solution $s' \in \mathcal{N}(s)$ is randomly chosen and accepted as new current solution depending on $f(s)$, $f(s')$ and T. The solution s' replaces s as new current solution if $f(s') < f(s)$ or, when $f(s') \geq f(s)$, with a probability which is a function of T, $f(s)$ and $f(s')$. Generally the probability is computed following the *Boltzmann distribution*. Metropolis et al. [31] have used this method when they simulated the movement of particles in cooled matter, therefore the name *Metropolis-Criterion* became popular for the following

```
procedure Simulated Annealing
    s ← GenerateInitialSolution()
    t ← 0
    T ← T_0
    repeat
        repeat
            s' ← arbitrary solution ∈ N(s)
            if f(s') < f(s) then
                s ← s'
            else
                if Z < e^{-|f(x')-f(x)|/T} then
                    s ← s'
                end if
            end if
            t ← t + 1
        until temperature-update conditions met
        T ← g(T,t)
    until termination conditions met
end procedure
```

Algorithm 2: Simulated Annealing

inequation:

$$\begin{aligned} Z &< e^{-|f(x')-f(x)|/T} \\ \text{with } Z &= \textit{random number} \in [0,1) \end{aligned} \tag{3.1}$$

The most crucial part in parameterizing simulated annealing is the selection of an appropriate *cooling scheme*, which strongly affects convergence speed and result quality. The idea is to decrease temperature during the search process so that at the beginning uphill moves are accepted with a high probability which decreases step-by-step with the following iterations. This is analogous to the natural process of annealing metals or glass.

While temperature is relatively high the search is not biased in a strong way and uphill moves are accepted regularly, with descending temperature the search is biased towards classical iterative improvement and accepting uphill moves will become unlikely; Simulated annealing can therefore be understood as a mixture of a random walk and iterative improvement.

The cooling scheme defines the temperature T at each iteration t of the annealing process. It consists of the definition of a starting temperature T_0, a function $g(T,t)$ with which the actual cooling is computed and the number of iterations between updates of the temperature. The choice of T_0 can be made upon statistical data or bounds. The number of iterations at each temperature should allow the procedure

to reach a stable state, which means that no more moves that only are allowed at this temperature should be necessary to reach a global optimal solution — physicists call this state an equilibrium. This number of iterations is usually set to a multiple of the size of the neighborhood. For updating the temperature no specific type of function is necessary, but commonly a monotone descend function is used, e.g. geometric cooling.

$$\begin{aligned} g(T,t) &= T \cdot \alpha \\ \text{s.t. } \alpha &< 1 \end{aligned} \tag{3.2}$$

The advantages of simulated annealing are that it is one of the best studied metaheuristics existing. For example it is proven that under certain conditions, e.g. infinite runtime, simulated annealing converges to a global optimum (Henderson and Jacobson [21]). Simulated annealing is easy to implement and can be adopted to a wide range of applications, although for good results often long runtimes are needed.

Simulated annealing is subject of continued research. Some of the more recent trends to improve practical performance are advanced cooling schemes including non-monotonic cooling (*reheating*), dynamic cooling, deterministic neighborhood exploration, parallelization and hybridization with for example genetic algorithms or GRASP.

3.3 Tabu Search

The elementary ideas of *tabu search* (TS) were first introduced by Glover [16] in 1986. Tabu search is one of the most cited and applied metaheuristics in the field of combinatorial optimization problems. In its basic version, described in Algorithm 3 on the next page, tabu search performs a best improvement local search (see Section 3.1 on page 19) and additionally uses a *short term memory*, which allows to escape from local optima and avoids cycles during exploration of the search space. This short term memory is implemented as a *tabu list* that remembers recently visited solutions and forbids moves towards them. The neighborhood of the current solution is restricted to solutions that do not belong to the tabu list, the resulting set is the so-called *allowed set*.

Similar to other metaheuristic methods an initial solution is generated randomly or with a construction heuristic, the tabu list TL is initialized with the empty set. At each iteration of the search process the best solution of the *allowed set* of the neighborhood of the current solution is selected as new current solution and added to the tabu list; an element of the tabu list is removed from it; usually the selection of this element is based on recency, i.e. removal in FIFO order. An essential property

```
procedure Basic Tabu Search
    s ← GenerateInitialSolution()
    x ← s
    TL ← ∅
    repeat
        X' ← part of 𝒩(x) that does not violate TL
        x' ← best solution ∈ X'
        add x' to TL
        remove elements older than t_L iterations from TL
        x ← x'
        if f(x) < f(s) then
            s ← x
        end if
    until termination conditions met
end procedure
```

Algorithm 3: Basic Tabu Search

of this process is that it allows to select new solutions with a worse solution quality than the current solution, because the search must not stop when it finds the first local optimum.

An important parameter is the length of the tabu list (*tabu tenure*). Small tabu tenures allow the process to concentrate on small areas of the search space. On the other side, large tabu tenures will forbid the process to revisit more solutions and thus a better exploration of the entire search space is enforced. The tabu tenure can be varied during the search process to improve the robustness of the algorithm and quality of results. Robust tabu search (see Taillard [41]) changes the tabu list length randomly during the search between a mininum and maximum size, while reactive tabu search (see Battiti and Tecchiolli [3]) increases the tabu tenure if there is evidence that some solutions are visited repeatedly. As a result the diversification of the process is increased, while the tabu tenure is decreased if there is no further improvement, which intensifies the search process.

However, the major problem of this basic tabu search algorithm is that it stores complete solutions in its short term memory. Managing tabu lists is thus inefficient because they make exhaustive use of memory and it takes significant computational effort to deal with them. Therefore, instead of storing complete solutions only *tabu attributes* are typically stored. These attributes characterize a performed move.

E.g. in case of the traveling salesman problem when a 2-opt move is performed the two removed edges or alternatively the two newly introduced edged may be stored as tabu attributes, and every solution that is generated using this attributes does not qualify for the allowed set, it is *tabu*. Because more than one attribute can be defined, a tabu list is introduced for each of these attributes.

This new type of tabu lists is much more effective, although it raises a new problem. With forbidding an attribute as tabu, typically more than one solution is declared as tabu. Some of these solutions that must now be avoided might be of excellent quality and have not yet been visited. To overcome this problem, *aspiration criteria* are introduced which allow to override the tabu state of a solution and thus include it in the allowed set. A commonly used aspiration criterion is to allow solutions which are better than the currently best known solution. A sketch of the procedure summarizing the above techniques is provided in Algorithm 4.

procedure TABU SEARCH
  $s \leftarrow$ GenerateInitialSolution()
  $x \leftarrow s$
  $TL_1 \dots TL_n \leftarrow \emptyset$
  repeat
    $X' \leftarrow$ part of $\mathcal{N}(x)$ that does not violate $TL_1 \dots TL_n$ or satisfies at least one aspiration criterion
    $x' \leftarrow$ best solution $\in X'$
    add x' to $TL_1 \dots TL_n$
    remove elements older than t_L iterations from $TL_1 \dots TL_n$
    $x \leftarrow x'$
    if $f(x) < f(s)$ **then**
      $s \leftarrow x$
    end if
  until termination conditions met
end procedure

Algorithm 4: Tabu Search

Additionally to the above described tabu lists, which represent a short term memory, other ways of taking advantage from information about the search history are possible. Every piece of information collected during the search process can be useful. This *long term memory* can be structured regarding to four principles: *recency*, *frequency*, *quality* and *influence*. A recency-based memory records for each solution, or attribute, the most recent iteration it was considered in, while frequency-based memory counts how many times each solution (attribute) has been visited. This information identifies the subset of the search space where the process stayed for a longer number of iterations or where it only examined a limited amount of solutions, so it is useful to control the diversification of the search process. The information regarding quality can be used to determine good solution attributes, which can be integrated in solution construction. Finally influence, a property regarding decisions during the search process, allows to identify the most critical decisions.

For further information the reader is encouraged to look at two articles by Fred Glover [17, 18], which provide a good starting-point for deeper insight into tabu search and related methods.

3.4 Guided Local Search

Guided local search (GLS) is a metaheuristic that sits on top of another local search procedure. It modifies the landscape of the search space to guide the underlying heuristic method away from already encountered local optima. The roots of the GLS metaheuristic are in a neural-network based method called GENET (see Tsang and Wang [43]) which is a constraint satisfaction resolution method.

As mentioned, GLS modifies the landscape of the search space, to guide the underlying local search method gradually away from known local optima. To accomplish this it augments the objective function of the underlying local search procedure with penalties, which makes the known local optima *less attractive* (see Figure 3.1). In Algorithm 5 on page 27 the basic guided local search procedure is described in pseudocode.

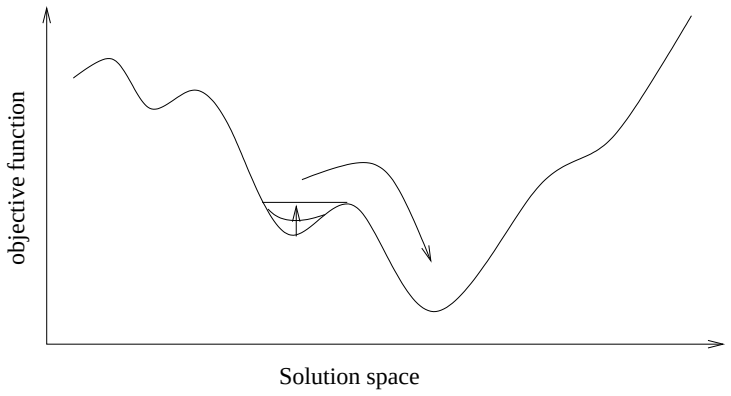


Figure 3.1: Escaping a local optimum with GLS

Guided local search applies the penalties to solution features which have to be defined. These features may be any property or characteristic that can be used to distinguish solutions; compare the tabu attributes of tabu search. E.g. in the case of the traveling salesman problem these features could be arcs between pairs of cities and in the case of the quadratic assignment problem facility-location assigments (see Voudouris and Tsang [44] and Mills et al. [32]). For each defined feature f_i the following components must be provided:

- An *indicator function* $I_i(s)$ that indicates whether the feature f_i is present in the current solution or not.

$$I_i(s) \quad = \quad \begin{cases} 1, & \text{solution } s \text{ exhibits feature } i \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

- A *cost function* $c_i(s)$ describes the cost of having the feature f_i present in the

current solution s. These costs are often defined in analogy to the objective function.

- And finally p_i, the *penalty parameters*, which are initialized with 0 for all features. The penalty parameters are used to penalize features that appear in local optima.

Given an objective function $g(s)$, which maps each solution of the search space to a numeric value, GLS defines a new augmented objective function $h(s)$ which will be used by the underlying local search procedure.

$$h(s) \quad = \quad g(s) + \lambda \cdot \sum_{i=1}^{n} I_i(s) \cdot p_i \tag{3.4}$$

Updating the penalty values p_i of the features when reaching a local optimum is the crucial task in guided local search. A common way to do this is to calculate a utility value $Util(s, i)$ of a feature i at the current local optimum s:

$$Util(s, i) \quad = \quad I_i(s) \cdot \frac{c_i(s)}{1 + p_i} \tag{3.5}$$

The penalty values of the features with maximimum utility value will be incremented. Then, local search is applied again with the updated penalties and changed augmented objective function.

The higher the costs $c_i(s)$ the higher the utility of the feature. The costs are scaled by the penalty value to permit the search process from being totally cost driven by taking the search history into account. A problem is that during the search process, where more and more features are penalized, the landscape of the search space could be distorted too much. This will make further exploration difficult and so in addition to increasing the penalty values a multiplication rule is applied regularly, which is smoothing the landscape again.

The λ parameter, also called *regularization parameter*, is used to specify the influence of the penalty values on the augmented objective function, which controls the diversity of the search process. With increasing λ the diversification will increase, too. The right choice of λ is crucial. This, however, must be done individually for each problem, because it is specific to the used objective function $g(s)$. The difference Δh of the values of the augmented objective function between two consecutive moves helps to understand this.

$$\Delta h \quad = \quad \Delta g + \lambda \cdot \sum_{i=1}^{n} \Delta I_i \cdot p_i \tag{3.6}$$

If the regularization parameter λ is large enough the inner local search procedure will solely remove the penalized features and the information regarding penalty values will fully determine the path of the search process. In contrast if, λ is to small the local search procedure will ignore the penalty values and it will not be able to escape from local optima. A good choice of λ is therefore in the same order of magnitude as Δg and the resulting moves will aim at the combined objective, which is to improve the solution and to remove penalized features from the generated solutions. A common solution for this problem is to introduce a α parameter which is used to tune the now dynamically computed λ parameter, taking into account information about the problem instance. The advantage of this method is that once α is tuned well enough it can be used for many problem instances (see Voudouris and Tsang [44]).

procedure GUIDED LOCAL SEARCH
  $s \leftarrow$ GenerateInitialSolution()
  for $i = 1, \ldots, n$ **do**
    $p_i \leftarrow 0$
  end for
  repeat
    $s \leftarrow$ LocalSearch(s,$g + \lambda \cdot \sum_{i=1}^{n} I_i \cdot p_i$)
    for all features i with maximum utility $Util(s, i)$ **do**
      $p_i \leftarrow p_i + 1$
    end for
  until termination conditions met
end procedure

Algorithm 5: Guided Local Search

Here only the main concepts of guided local search are described but many additional ideas and improvements where proposed and applied successfully in different applications such as Fast GLS. Also several other refinements of the algorithm are possible such as e.g. iterative penalty value updates (Voudouris and Tsang [42] and [45]).

3.5 Greedy Randomized Adaptive Search Procedure

The *Greedy Randomized Adaptive Search Procedure* (see Feo and Resende [11, 12]) is a simple but powerful metaheuristic that combines a constructive heuristic with local search. The basic structure of GRASP is outlined in Algorithm 6 on the following page. GRASP is an iterative multi-start procedure which consists of two phases, the construction phase builds a feasible solution, whose neighborhood is explored to

find a local optimum in the subsequent local search phase. The best solution found in any iteration is returned as final result of the search process.

```
procedure GreedyRandomizedAdaptiveSearchProcedure
    repeat
        s' ← GreedyConstructSolution()
        s' ← LocalSearch(s')
        if f(s') < f(s) then
            s ← s'
        end if
    until termination conditions met
end procedure
```

Algorithm 6: Greedy Randomized Adaptive Search Procedure

The construction phase itself, outlined in Algorithm 7 on the next page, is characterized by two major properties: a dynamic constructive heuristic and randomization. It is assumed that a solution consists of a subset of components, analogous to Section 3.4 on page 25 where these components could be used as GLS features. During the construction phase the solution is put together step-by-step, adding a new component in each iteration. The selection of the new component is done at random out of the *restricted candidate list* (RCL). It is essential that the construction heuristic is *dynamic*, which means that the score for each solution component is evaluated depending on the current partial solution. In contrast *static* construction heuristics assign a score to each solution component prior to the construction process.

The most critical part of the GRASP construction phase is the *BuildRestrictedCandidateList* procedure, since it determines the strength of the heuristic bias. An incremental cost $c(e)$ is associated with the inclusion of a component $e \in CL$ into the currently constructed solution. Further at each iteration let c_{min} and c_{max} be the smallest and the largest incremental costs and subsequently the restricted candidate list is made up by the most promising components $e \in CL$, i.e. with the best incremental costs $c(e)$.

An easy solution for this problem is to limit the RCL by the number of its elements. The list is made up by k components with the best incremental costs $c(e)$, where k is a parameter which has to be carefully tuned. In its extremes k is either set equal to 1, resulting in a construction procedure which degenerates to a deterministic greedy heuristic, because only the best element at each iteration would be considered for the RCL. If $k = n$, where n is the size of CL, i.e the number of possible components, the construction is done completely at random.

On the other side the restricted candidate list can be limited by the quality of the components. Therefore a threshold parameter $\alpha \in [0, 1]$ is associated with the RCL.

```
procedure GreedyConstructSolution
    s ← ∅
    while solution s is not complete do
        CL ← all possible extensions e of solution s
        RCL ←BuildRestrictedCandidateList(CL)
        e ← select an element of RCL at random
        s ← s ⊕ e
    end while
end procedure
```

Algorithm 7: GRASP construction phase

All components whose costs $c(e)$ are superior to the threshold value are included, so the condition $c(e) \in [c_{min}, c_{min} + \alpha \cdot (c_{max} - c_{min})]$ has to be fulfilled by each element of the RCL. In analogy to the previous RCL selection method the extreme cases exist, too, with $\alpha = 1$ resulting in a pure greedy heuristic and $\alpha = 0$ equivalent to a pure random construction.

In both cases k and respectively α are important parameters which strongly determine the sampling of the search space and hence the quality of the resulting solutions. It is essential to the success of GRASP that the most promising regions of the solution space are sampled during the construction phase. Also it is important that the constructed solutions belong to basins of attraction of different local optima to ensure sufficient diversification. The first condition can be achieved by a good choice of the construction heuristic and its parameters. For the second condition an appropriate choice of the construction heuristic and the subsequent local search are the key to success.

In the given description of the GRASP metaheuristic memory in terms of history was not mentioned. This is one of the reasons why GRASP is often outperformed by other metaheuristics. However, due to its simple concept GRASP is easy to implement for many applications. For example, applications exist for the set covering and maximum independent set problem by Feo and Resende [12] or the quadratic assignment problem by Li et al. [26]. Also the iterations for creating candidate solutions usually are fast and so GRASP is able to provide good quality solutions in a short amount of time.

To improve the performance of GRASP several techniques are possible. As mentioned above the construction phase, especially the creation of the restricted candidate list, is critical. Some enhancements address this problem. With Reactive GRASP the RCL parameter α is not constant; in each iteration it is selected from a discrete sequence [37], yielding in a more robust algorithm. Other methods include a biased selection of new elements from the RCL, e.g. with a probability proportional to $1/c(e)$. Parallelization can also be easily applied to GRASP [38].

Current research trends show that GRASP can gain a great performance boost if it is used in a hybrid manner. So it is possible to use greedy constructed solutions as starting population within evolutionary algorithms. The use of simulated annealing or tabu search within GRASP has been applied successfully, too.

Not even the gods fight against necessity.

Simonides

Chapter 4

Requirements

At the beginning of this master thesis the basic idea was to extend the existing *EAlib* [35] with some additional metaheuristics, since EAlib at that time only contained evolutionary algorithms. The EAlib is intended to be a problem-independent C++ library suiteable for the development of efficient metaheuristics for combinatorial optimization. It is developed at the Institute of Computergraphics and Algorithms, Vienna University of Technology, Austria since 1999.

This chapter is structured into a description of the *functional*, *design* and *usability* requirements that were stated initially.

4.1 Functionality

Before we start off, a summary of the functionality that EAlib provided already is given. As mentioned, EAlib included initially classes for evolutionary algorithms (EA). In particular classes that provide a generic framework for a generational EA, a steady state EA and an EA using the island model were implemented. Some supporting classes designed for populations and subpopulations, chromosomes, i.e. solutions, parameter handling and logging were provided, too. For demonstration purposes an implementation of the simple ONEMAX problem is also included.

As mentioned the primary goal is to enhance EAlib with classes that provide a framework for some commonly known metaheuristics. After some consideration we selected the following five:

- Local Search
- Simulated Annealing
- Tabu Search
- Guided Local Search

- Greedy Randomized Adaptive Search Procedure

Additionally an auxiliary more complex example problem should be implemented, for which we chose the already described quadratic assignment problem. Another important task is to enhance the existing parameter handling mechanism, because EAlib initially only featured a global parameter namespace.

In the following we describe in detail the functional requirements on the individual components of the implementation.

Local Search

An iterative improvement algorithm as described in Section 3.1 on page 19 should be developed. Therefore the standard step functions random neighbor, next improvement and best improvement are required too.

Simulated Annealing

The implementation of the simulated annealing algorithm should be straightforward. It should feature geometric as standard scheme, and the acceptance probability of down hill moves is to be calculated with the Metropolis criterion.

Tabu Search

The main features desired for tabu search are handling of an arbitrary number of tabu lists for different purposes. Due to the requirement for tabu attributes are to be used, of course support of aspiration criteria must be provided too.

Guided Local Search

The requirements for the GLS implementation are straightforward. An appropriate mechanism for feature evaluation is needed. Additionally it is desired that the λ GLS parameter is automatically tuned, utilizing user provided α parameter and the size of the current instance, as described in Section 3.4 on page 25.

Greedy Randomized Adaptive Search Procedure

GRASP has not many requirements, a simple construction heuristic must be provided and the underlying local search algorithm should be selectable.

Example Problem

Besides the actual implementation of the generic algorithms an example problem has to be addressed. It serves two different purposes, at first it should of course show

the possible potential of the used metaheuristics and secondly it should act as a template for developing other applications with EAlib. But of course demonstrating the benefits of a generic implementation of metaheuristic algorithms, like we did in this master thesis, is also one of the aims to be achieved.

To fulfill this requirements certain aspects have to be considered:

- it must be a *combinatorial optimization problem*, since EAlib is designed for this type of problems,
- computational and pratical *hard* to solve
- practical relevance of the problem
- existence of compareable results
- existence of *standard* instancances for testing purposes
- well known
- easy understandable problem structure
- adequate to fulfill demonstration purposes

Initially we considered three proplems, maximum satisfiability, quadratic assignment and glass cutting. The latter one was droppen early because it is too complex for use as a demonstraton problem. As noted, finally the quadratic assignment problem was selected.

In particular the QAP implementation must feature all algorithms with their specialities. I.e. appropriate step functions, tabu attributes, features for guided local search and a construction heuristic are needed.

Parameter Handling

The initial version of EAlib only featured one global parameter namespace in an application. Although this concept is simple and robust the major drawback of it is, that hierarchical parameter settings are not possible.

This is not satisfactory when for example nested algorithms, like guided local search or GRASP which incorporate another *inner local search algorithm*, or other advanced methods are used. It is obvious that the inner local search should be parametrised without tampering the parameter settings of the outer algorithm.

Apparently the extended parameter handling has to ensure compatibility with existing applications.

4.2 Design

Building a problem independent library is a complex task and many decisions are not obvious at hand. Therefore designing such a library is a sophisticated task to acccomplish. Though this master thesis is based on an existing library and so many decisions are somewhat constrainted.

Special attention has to paid for the design of the specialities of the individual algorithms, because they should not interfer each another, but as much as possible of the original ideas should be realised. To accomplish this special functionality is to be declared in a separate *interface class* which must be inherited if a class implements it. Examples for such interfaces are:

- augmented objective values
- construction heuristics
- features
- tabus
- tabulists

The use of common coding patterns is also encouraged, to make live easier for future changes and enhancements and to help developers understanding the source-code. For example functionality should be divided in reasonable methods within a particular class to ease customizations by users.

4.3 Usability

The EAlib is designed to help developing metaheuristics for combinatorial optimization problems. Therefore it is important that the user-visible part of the desired EAlib extensions meet some fundamental requirements which are summarised here:

- easy to learn and clear programming interface
- good documentation, at best with an C++ language integrated tool like *doxygen*
- support for basic features included

Work saves us from three great evils: boredom, vice and need.

Voltaire

Chapter 5

Implementation

In this chapter a detailed description of our implementation is provided. At first a in-depth look on the internals is given in Section 5.2 on page 37 and afterwards an usercentric description of the interface and a guide for using the new classes is give in Section 5.3 on page 46.

5.1 Overview

In Figure 5.1 on the next page an instant overview of the most important EAlib classes is given in UML syntax. The figure shows the how the classes are related to each other by inheritance, realization or an other dependency.

As depicted in the EAlib class overview the most important base classes are *ea_base* and *chromosome*, where ea_base is the top level base class for all algorithms, for both evolutionary and local search alike algorithms and chromosome is the top level base class for user provided problems, i.e. if a new application is to be created using EAlib it is required that the actual problem is implemented as a child class of chromosome and if necessary of some interfaces for to provide specialized methods for certain algorithms, e.g. guided local search.

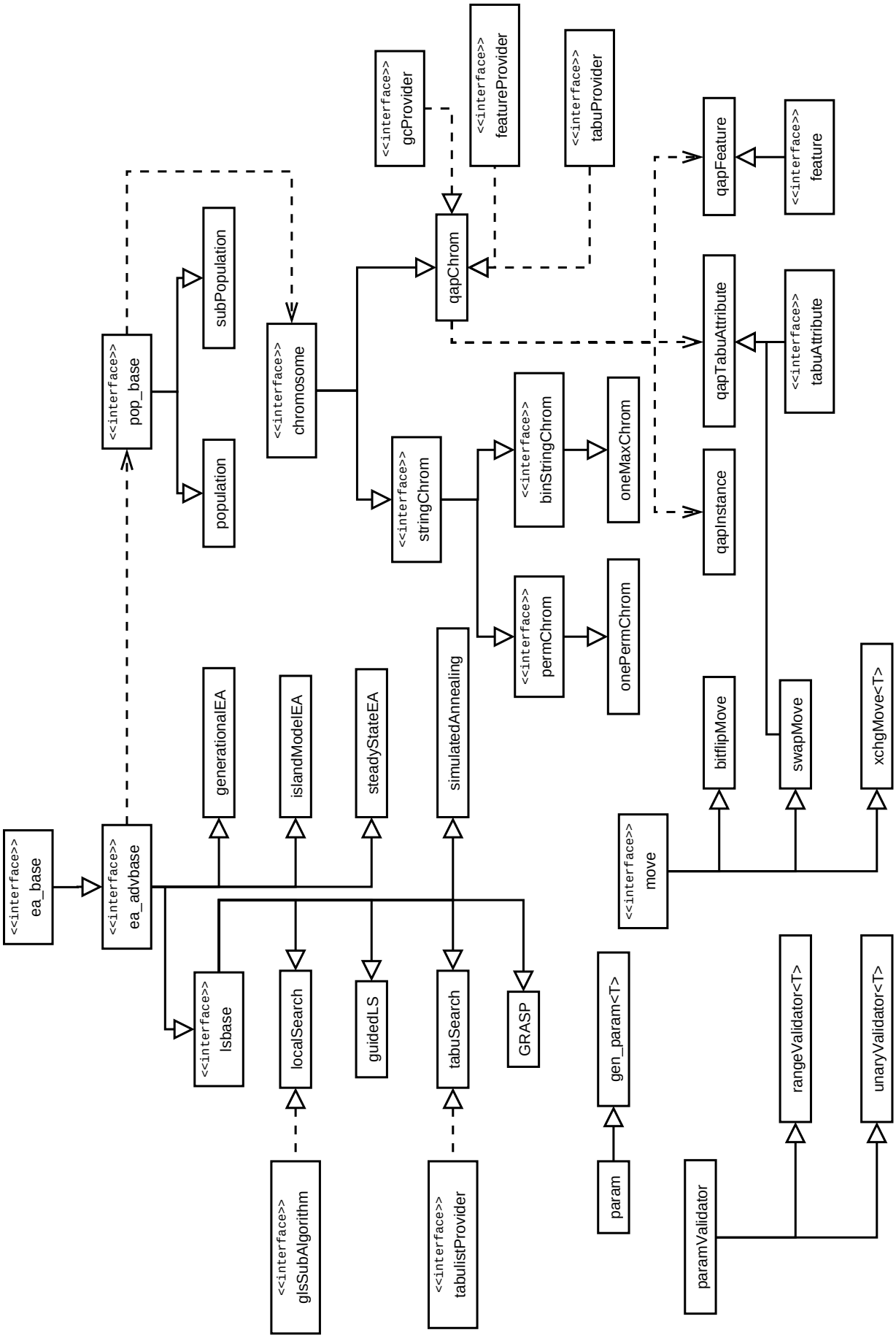


Figure 5.1: EAlib class overview

5.2 Class reference

In this section we present the details of the implementation at hand. It is structured into a description of the individual classes and of the developers view of the new parameter handling mechanism.

5.2.1 Class chromosome

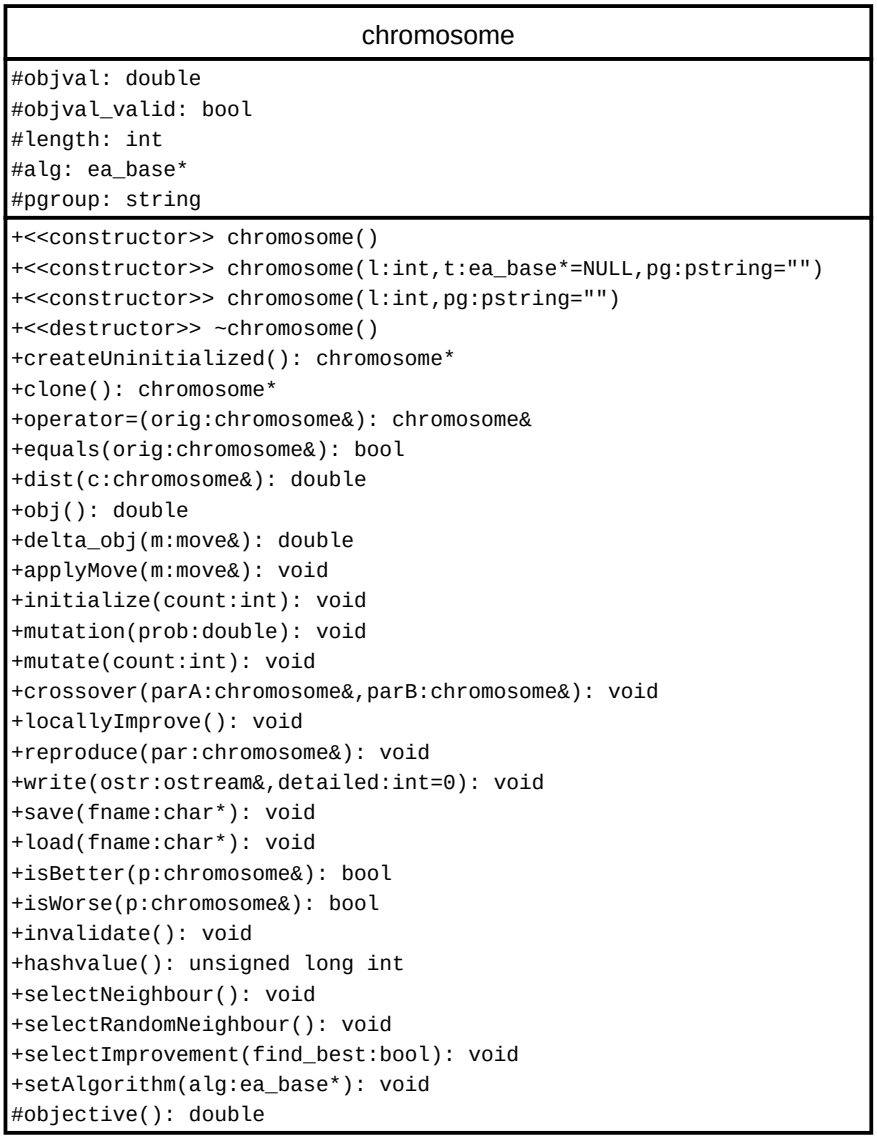


Figure 5.2: Class chromosome

This is the base class for all chromosomes. In EAlib the problem definition is given by deriving a new class from *chromosome* and implement all the problem relevant methods in a proper way, i.e. all pure virtual methods have to be implemented. additionally depending on the desired use, other virtual methods have to be reim-

plemented, e.g. *save* and *load* or *selectRandomNeighbour* and *selectImprovement* to enable local search alike algorithms.

5.2.2 Class ea_advbase

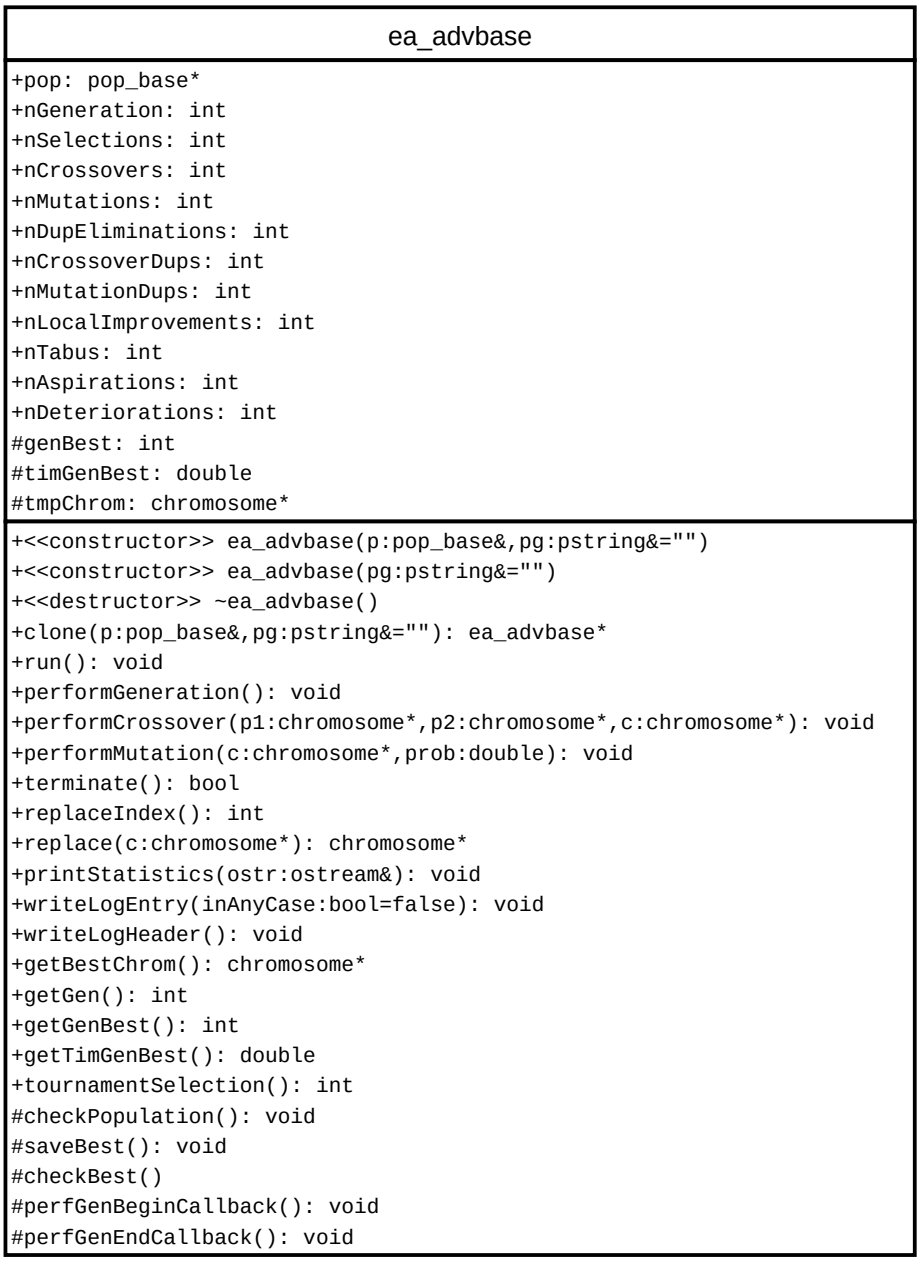


Figure 5.3: Class ea_advbase

The abstract base class for algorithms. Any new algorithm should use ea_advbase as the base class, if no other derived class suits the requirements.

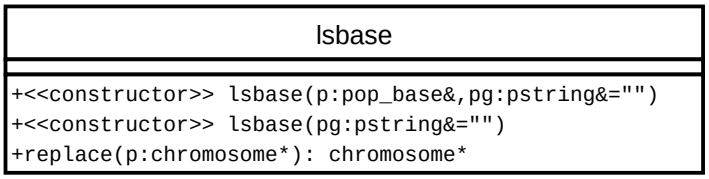


Figure 5.4: Class lsbase

5.2.3 Class lsbase

This is the base class for local search alike algorithms, i.e. algorithms that are not population base. To be as much compatible as possible with population based algorithms, no additional data members are introduced, instead a fixed subset of the population, namely the first element, is considered to be used.

5.2.4 Class localSearch

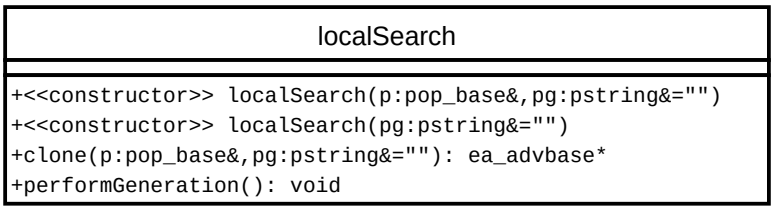


Figure 5.5: Class localSearch

The localSearch class implements the basic local search functionality as outlined in Section 3.1 on page 19. Additionally to its main base class *lsbase* it inherits the *glsSubAlgorithm* interface class, too, because we consider localSearch as embedded algorithm for guided local search.

5.2.5 Class simulatedAnnealing

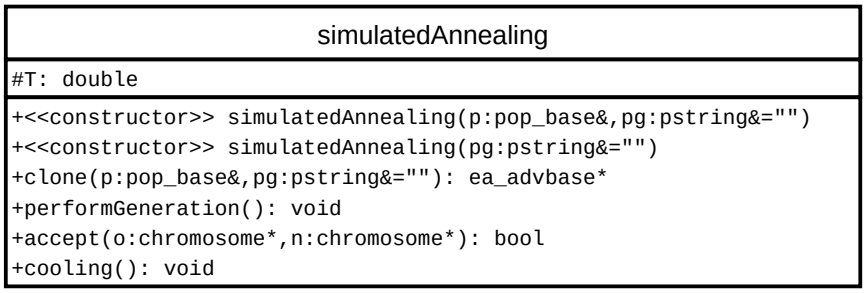


Figure 5.6: Class simulatedAnnealing

This class provides an application independent implementation of the simulated annealing metaheuristic (see Section 3.2 on page 20). The two main steps of the simulated annealing process are subdivided into the *accept* and the *cooling* methods. As suggested, the Metropolis-Criterion and geometric cooling are utilized. Compared with local search, simulated annealing adds a temperature to the state; therefor the attribute T is introduced.

In the overwritten version of the *performGeneration* method *accept* is called when a newly selected neighbour has an inferior objective value than the current solution. The *cooling* method is called accordingly to the *sacint* parameter.

5.2.6 Class tabuSearch

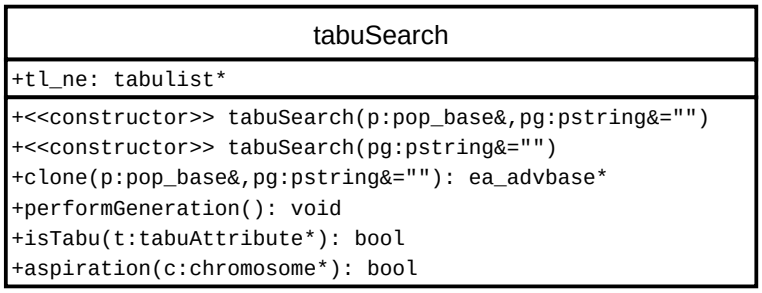


Figure 5.7: Class tabuSearch

The tabuSearch class offers a basic tabu search metaheuristic (see Section 3.3 on page 22). It utilizes a single tabulist and contained tabu attributes are considered as tabu. To overcome the problem of prohibiting a solution that is the best known sofar, a default aspiration criterion is implemented, too.

Chromoses used in combination with tabu search have to inherit the *tabuProvider* interface class and should therefore take care of the embodied tabulists appropriately.

5.2.7 Class guidedLS

This class provides a guided local search algorithm (see Section 3.4 on page 25), which can only be used in combination with a chromosome class that implements the *featureProvider* interface; the embedded algorithm has to be derived from *glsSubAlgorithm* class. This is necessary to ensure all involved classes are well prepared with respect to the requirements of guided local search.

Statistics of the embedded algorithms are summarized, whereas the generation counter is threated individually to avoid sideeffects related to termination criteria and penalty resets.

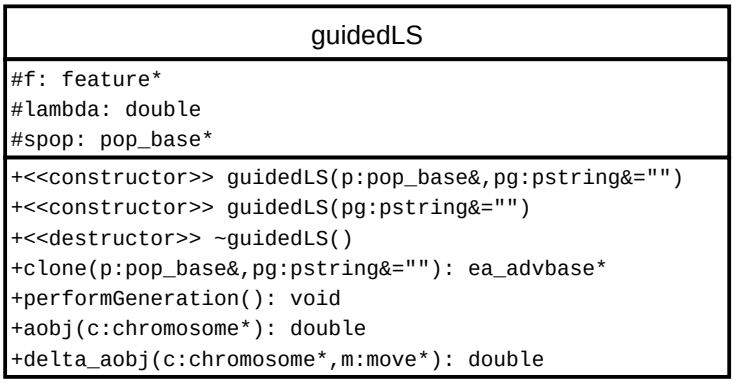


Figure 5.8: Class guidedLS

To provide a population for the embedded algorithm an additional population is created by using the existing chromosomes as a template. This ensures that running the embedding algorithm has no hidden side-effects.

5.2.8 Class GRASP

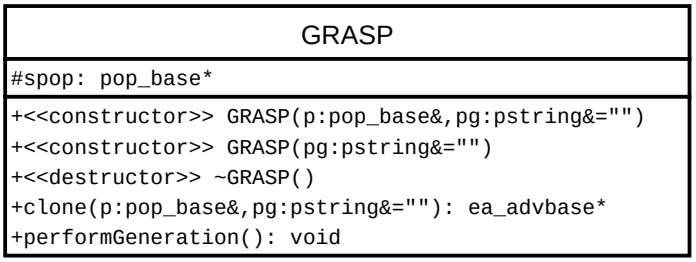


Figure 5.9: Class GRASP

The GRASP class implements the greedy randomized adaptive search procedure metaheuristic (see Section 3.5 on page 27). It can only use chromosome derviates that additionally implement the *gcProvider* interface, because the used chromsomes are required to provide a greedy construction heuristic.

Analogous to guided local search an additional population is created to be used by the embedded algorithm to circumvent possible side-effects.

5.2.9 Class feature

As mentioned, the guided local search metaheuristic requires the definition of features, i.e. a method to identify if a feature is present in a given solution. This class is an abstract base for those problem dependent feature classes, that handle the identification of the features and their penalization. Although only one feature object is

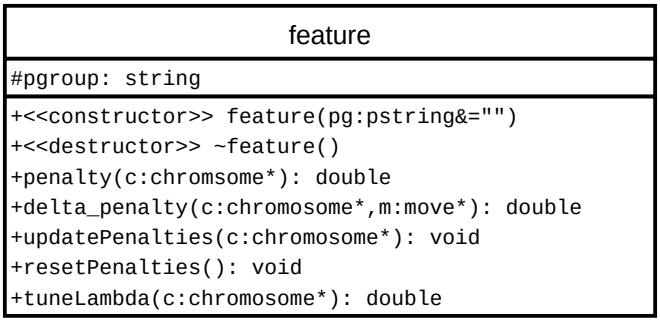


Figure 5.10: Class feature

used by the *guidedLS* class, it is possible to use several different types of features. However, due to this design the, it is very simple to access the features.

5.2.10 Class tabuAttribute

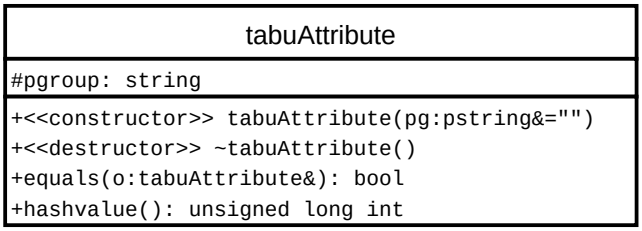


Figure 5.11: Class tabuAttribute

This abstract class provides an interface for classes whose objects are used as elements of a tabulist. It is important that the methods *equals* and *hashvalue* are implemented in a proper way, because they are invoked by an internal hashing array object of the tabulist, which requires this to methods. So tabus which should be considered matching need to return equal hashvalues and *true* for the equals method. Vice versa tabus which should not be considered as matching should at best return different hashvalues and *false* for the equals method. If equal hashvalues are return but the tabus are not matching this must be assured by the equals method. The *write* method is mainly used for debugging purposes during development.

5.2.11 Class tabulist

This class provides the basic tabulist functionality as used by the tabu search metaheuristic. It is implemented using a hashing array to ensure efficient matching of existing tabu attributes. An additional queue is utilized in order to memorize the insertion sequence of the tabu attributes into the tabulist. The hashing array is utilized by the *match* method to decide if a given tabu attribute matches any existing

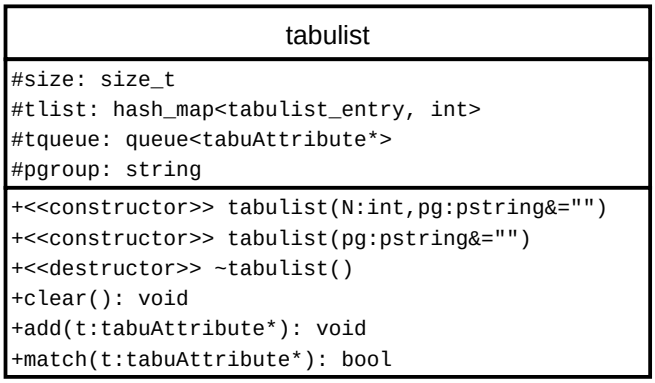


Figure 5.12: Class tabulist

tabu attribute. The queue is used to remove older elements from the tabulist as new elements are added by the search process.

5.2.12 Class move and childs

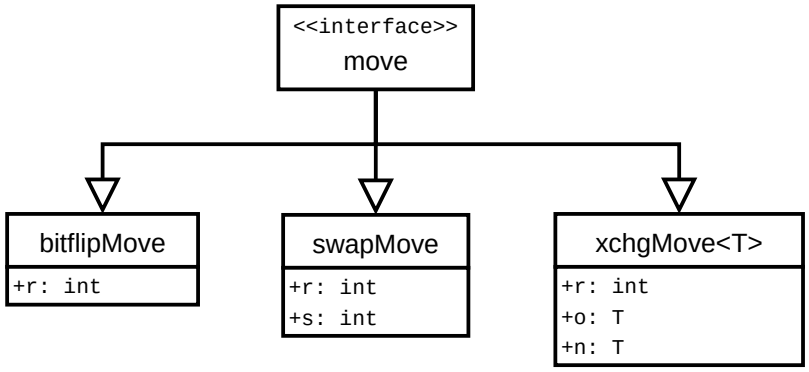


Figure 5.13: Class move and childs

This classes represent moves in the neighbourhood of a chromosome. Their objects are can be used for example within incremental update of the objective value or as a base for tabu attributes.

5.2.13 Class qapChrom

This is the main problem specific class which coordinates the interaction of all QAP related classes, i.e. *qapInstance*, *qapFeature* and *qapTabuAttribute*. It is derived directly from *chromosome* and its main member is a vector containing indicies of facilities, which represents a quaratic assignment.

Mutation is performed by swapping two elements of the solution vector. A cycle crossover is implemented, too. Due to efficiency concerns the static data of the ac-

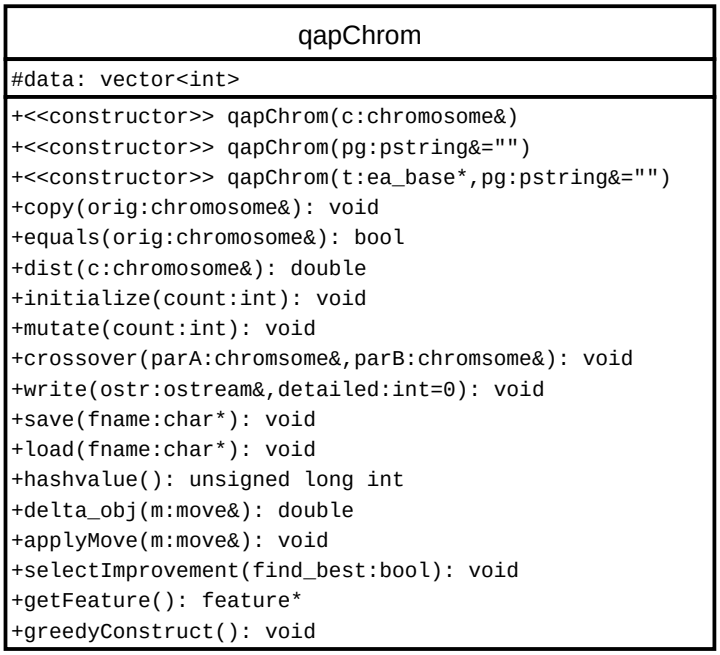


Figure 5.14: Class qapChrom

tual instance is stored in a global *qapInstance* object. To support each implemented algorithm (see Chapter 3 on page 18) the newly introduced interface classes *featureProvider*, *tabuProvider* and *gcProvider* are inherited and their virtual methods are implemented accordingly. In particular the classes *qapFeature* support guided local search and *qapTabuAttribute* tabu search. The construction heuristic proposed by Li, Resende and Pardalos [26] implemented, too.

5.2.14 Class qapInstance

An object of this class contains all necessary data for one particular quadratic assigment instance. It can load the instance data from a file whose filename is supplied as parametere to the constructor; if no filename is specified, the file to which the *qapfile* parameter is referring is loaded.

Auxilliary to the basic storage functionality the presorting stage of the construction heuristic used by *qapChrom* is done in this class for performance reasons. because the indices only need to be sorted once for an instance.

5.2.15 Class qapFeature

This is the specialized feature class for the quadratic assignment problem. The essential idea is that every possible facility-location pair is threated as a feature (see Section 3.4 on page 25). The penalties of these features can be efficiently stored in

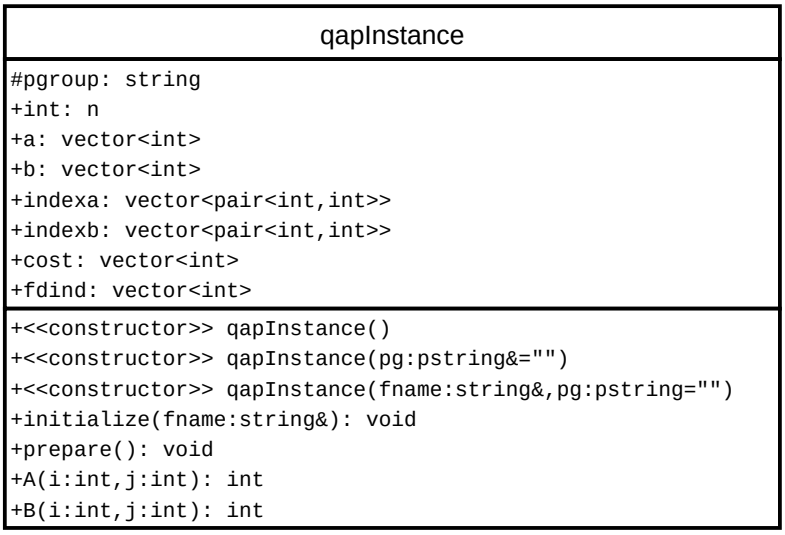


Figure 5.15: Class qapInstance

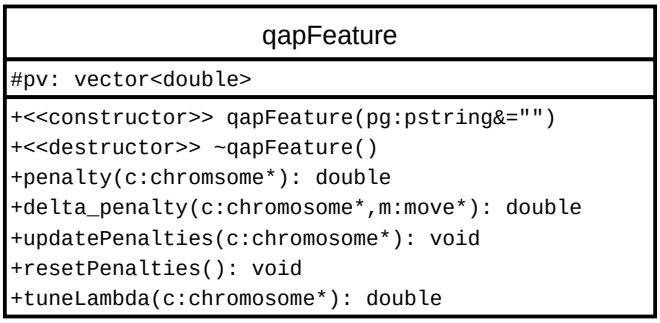


Figure 5.16: Class qapFeature

a two-dimensional matrix. To maintain the benefits of an incremental update of the objective value the change in penalty can be computed for a given move.

5.2.16 Class qapTabuAttribute

This tabu attribute is derived from the *swapMove* class. Two qapTabuAttributes are considered equal if the resulting chromosomes are equal when the moves they represent are applied to them.

5.2.17 Parameter handling

Originally the mechanism that EAlib used to deal with user provided parameter values had some disturbing deficiencies. At first it was not possible to handle multiple values for one parameter, which are for example distinguished by an additional parameter namespace or parameter key. This is especially a problem when one parameter is used in a different context at different places within EAlib, e.g. a guided local

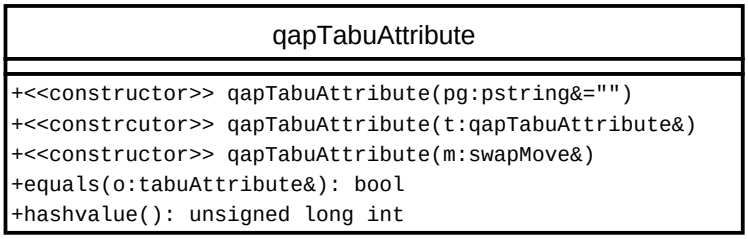


Figure 5.17: Class qapTabuAttribute

search metaheuristic uses an embedded simulated annealing metaheuristic and the two algorithms should use differently parametrized termination criteria. Secondly only one parameter validator class was implemented, which checks if a numeric value is in a given range.

As mentioned, a convenient solution for the first problem should maintain backwards compatibility. At this point the way how values of parameters are access in EAlib helps very much, because the operator () is used when a parameter value is access. This operator method can be changed so that it fulfills our requirements appropriately. In particular a string parameter is added to the operator () method, which defaults to an empty string representing the already existing global parameter namespace. When a different parameter key is specified the actual value is determined in the following order:

1. value associated with parameter key
2. global parameter value
3. default parameter value defined in the sourcecode

An efficent storage container for these parameter key and value pairs is provided by the hash_map class, which is an SGI/GNU extension to the C++ standard template library.

The second problem is solved by adding a validator which can perform an unary check such as greater or equal ($\geq$), greater ($>$), equal ($=$), less ($<$), less or equal ($\leq$) or not equal ($\neq$). Also the already existing range checking validator is extendend in way that bounds can be included or excluded from the allowed range.

5.3 Usage

In this section we give a summary on how to use EAlib with new optimization problems. As mentioned the problem description has to be incorporated in a new

class which is derived from the *chromosome* class or one of its already existing derviates depending which ever fits best the actual requirements.

However, inheriting class *chromosome* and implementing its pure virtual methods enables only the basic features. Therefore the new problem class is only useable by those algorithms that raise no specific requirements.

In EAlib this is solved by the introduction of interface classes for specific sets of features. The usage this interface classes is not limited to problem classes and hence it is used for algorithm classes, too. This concept has several advantages compared with collecting all methods in the class *chromosome*.

- The class *chromosome* is kept small and manageable,
- developers do not need to take care about features they are not interested in,
- new features can be integrated without affecting existing sourcecode.

If a class is providing the functionality of an interface class it has to be derived from it. The dynamic_cast operator can be used to determine if a certain class implements an interface.

In the remainder of this section the already existings interface classes are described.

5.3.1 Interface aObjProvider

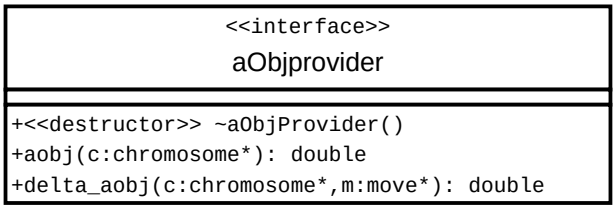


Figure 5.18: Interface aObjProvider

This interface class is to be inherited by algorithm classes that provide an additional term to the objective value of the chromosomes. The interface class *glsSubProvider* is derived from this class an should be used if the implemented algorithm should be used nested into guided local search.

5.3.2 Interface tabulistProvider

An algorithm class that provides tabulists has to inherit this interface class, to indicate the incorporated chromosome objects that they should use the tabulists.

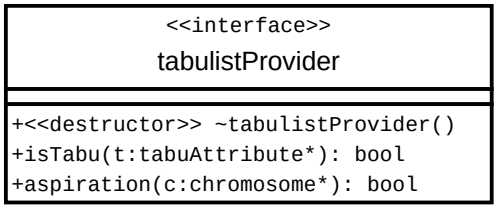


Figure 5.19: Interface tabulistProvider

5.3.3 Interface featureProvider

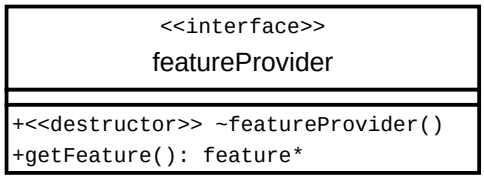


Figure 5.20: Interface featureProvider

Each chromosome class that is to be used in combination with guided local search has to inherit this interface class. An according class derived from class *feature* has to be realized, too.

5.3.4 Interface gcProvider

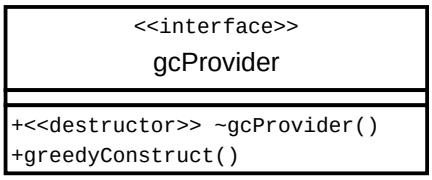


Figure 5.21: Interface gcProvider

If a chromosome class inherits this interface class, this indicates that a greedy construction heuristic is implemented within. This is mandatory if the greedy randomized adaptive search procedure metaheuristic is applied.

5.3.5 Interface tabuProvider

Chromosome classes which are able to deal with tabulists, that means they can fill them with tabu attributes and check if changes to them are currently tabu.

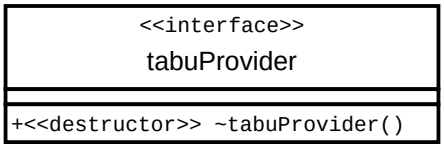


Figure 5.22: Interface tabuProvider

5.3.6 Parameters

As mentioned EAlib provides a powerful parameter handling feature. These parameters are used to configure the application. There exist parameters for a variety of domains, e.g.:

- algorithm configuration,
- problem specification,
- termination criteria,
- logging facility.

During this master thesis many parameters where added, some where changed in their semantics while others are only used. To illustrate what a user can customize a detailed overview of the parameters affecting this master thesis is given:

eamod The actual Algorithm to use. Currently the following choices are available:

- 0: steady-state EA,
- 1: generational EA,
- 2: steady-state EA with island model,
- 3: generational EA with island model,
- 4: simple randomized local search,
- 5: simulated annealing,
- 6: tabu search,
- 7: greedy randomized adaptive search procedure,
- 8: guided local search.

Default: 0

maxi Should be maximized? True if maximization, false for minimization.

Default: 1

mvnbop Neighbour selection function to use

- 0: random neighbour,
- 1: next improvement,
- 2: best improvement.

Default: 0

tgen The number of generations until termination.

Default: 100000

tcgen The number of generations for termination according to convergence.

Default: 0

tobj The objective value for termination when tcond==2.

Default: 0

ttime Specifies the amount of time the algorithm is allowed to run in user-space.

Default: 0

glsa Tuning parameter for the influence of penalties in guided local search.

Default: 0.5

glsri Interval of generations after which a penalty reset should be performed. If this value is 0 penalty resets will be disabled.

Default: 0

sacint Specifies the number of iterations between two successive cooling steps, in other words, the number of iterations for which the simulated annealing process stays at a certain temperature level.

Default: 1

satemp Specifies the starting temperature of the simulated annealing process.

Default: 1.0

tlsize Specifies the default size of newly created tabu lists.

Default: 10

qapfile This parameter specifies from which file the qapInstance object should read the actual instance data. The format of the problem data is the same as used by the QAPLIB [7] instances:

```
n
A
B
```

where n is the size of the instance, and A and B are either flow or distance matrix.

Default: “”

saca Specifies the slope for the geometric cooling of the simulated annealing process.

$g(T, t) = T * saca,\ 0 < saca < 1$

Default: 0.95

graspa Alpha parameter for GRASP. It is used in the both stages of the QAP construction heuristic and controls the candidate restriction.

Default: 0.5

graspb Beta parameter for GRASP. It is used in the first stage of the QAP construction heuristic and controls the candidate restriction.

Default: 0.1

By far the best proof is experience.

Sir Francis Bacon

Chapter 6

Experimental Results

To be able to compare the results from this different methods many internet available QAP instances are used, the probably most important collection of instances is the QAPLIB [7].

Of special interest are problem instances with a known optimal solution, especially if they are of larger size. To address this requirement some algorithms for construction of such instances where proposed, for example the by Palubeckis [34].

6.1 Test Cases

Because the larger quadratic assignment problem instances are computationally intractable, suboptimal algorithms such as the previous mentioned heuristics and metaheuristics are very popular and enjoy wide use. However, when dealing with new methods the quality and other properties such as robustness are important to know before they can be applied in daily business or other critical environments.

Usually new algorithms are tested on QAP instances from the QAPLIB (see Burkard, Karisch and Rendl [7]) which is a public internet available collection of well known instances which allows to compare algorithms with each other. However, the problem when using especially larger benchmark problems from QAPLIB is that the optimal solutions are not known in general and one has to rely on lower bounds (see Section 2.3 on page 11).

To overcome this problem an other set of test cases can be used. Those are generated by special algorithms whose output are not only the matrices which the define the problem but also with a provable known optimal solution. It has been shown that instances generated by such algorithms are rather hard to solve for some metaheuristics, namely simulated annealing, tabu search and others [34].

6.2 Test Setup and Procedure

Our tests were performed on an ordinary desktop computer with GNU Linux installed. The key data of this testing system is listed below:

CPU	Intel Pentium 4 2.8 GHz
OS	Linux 2.4.21 GNU Libc 2.3.2
Compiler	GCC 3.3.1 binutils 2.14.90.0.5

Table 6.1: System setup

We compiled EAlib and our test application for the quadratic assignment problem with all documented speed optimizations enabled, i.e. with switch -O4.

The test instances are all included in the already mentioned QAPLIB problem library. Each algorithm had to solve each instance 25 times, whereat each run had a time limit of five minutes to complete. The parameter settings for the particular instances were made upon our knowledge which we obtained during preceeding experiments.

In the following tables the parameter settings for each algorithm are given which were not at their default value.

Parameter	Value
maxi	0
ttime	300
tgen	0
tcgen	5000

Table 6.2: Parameter settings for Local Search

During the testruns of local search the *tcgen* parameter was set so that the search process could terminate if now improvement was made for 5000 iterations, which occurred quite of often. However, this parameter setting did not affect the achieved results significantly.

Parameter	Value
maxi	0
ttime	300
tgen	0

Table 6.3: Parameter settings for Simulated Annealing

For simulated annealing the parameters controlling the temperature schedule were set to estimated values dependig on the size of the problem instance to solve and the order of magnitude of the corresponding objective value.

Parameter	Value
maxi	0
mvnbop	2
ttime	300
tgen	0

Table 6.4: Parameter settings for Tabu Search

The parameter *tlsize* which controls the length of the tabulist was set depending on the actual problem instance. The value chosen was in order of magnitude of the size of instance to solve.

Parameter	Value
glsri	5000
maxi	0
ttime	300
tgen	0
sub.eamod	4
sub.ttime	0
sub.tcgen	500

Table 6.5: Parameter settings for Guided Local Search

Parameter	Value
maxi	0
ttime	300
tgen	0
sub.eamod	4
sub.ttime	0
sub.tcgen	500

Table 6.6: Parameter settings for GRASP

Additionally the *tobj* parameter has been set accordingly so that each testrun at which global optimum was found was terminated as soon as this global optimum was reached. This parameter setting did not change the results but sped up the experiments significant.

6.3 Results

For the comparison of the individual algorithms which were implemented and interpretation of the results four different charaterisic values are used, which in combination give a good insight into the data obtained during the experiments.

- *count of reached optima* per problem instance. It is a measure for stability of the search process. For the overall statistics the sum of the particular instances is used,
- *best objective value* per problem instance indicates primarily the potential quality of the search process,
- *mean objective value* is a measure for both quality and stability of an algorithm. However, outliers can have great influence on its value,
- *deviation of objective value* indicates the robustness of the search process.

Obviously the latter three indicators can not be compared directly among different problem instances. Therefore they are presented in %-gap notation relative to the known global optimal solution. The %-gap of an objective value x relative to the global optimum *opt* is calculated as follows:

$$\%-\text{gap}(x, opt) = \frac{x - opt}{opt} \times 100\% \tag{6.1}$$

At first table 6.7 and accordingly figures 6.1 and 6.2 show the overall results of each algorithm for all test instances together, i.e. the sum of the number of reached optimas and the mean %-gap across all testruns. With the obtained results no definitive winning algorithm can be declared. Nevertheless the results show clearly which of the implemented methods are well suited to solve quadratic assignment problems.

Algorithm	# Opt	Mean %-gap
Local Search	2	9.22
Simulated Annealing	132	3.47
Tabu Search	186	0.91
Guided Local Search	494	0.29
GRASP	485	0.12

Table 6.7: Overall results

Looking at the detailed results presented in follwing tables (6.8, 6.9, 6.10, 6.11 and 6.12) show that guided local search and GRASP are indeed clearly outperforming the other algorithms.

This is not very surprising because both guided local search and GRASP include received considerable more problem dependent knowledge in their implementation than the other metaheuristics implemented during this master thesis.

It is also worthy to mention that guided local search and GRASP were the only algorithms which were able to find distinct optimal solutions if the problem instance

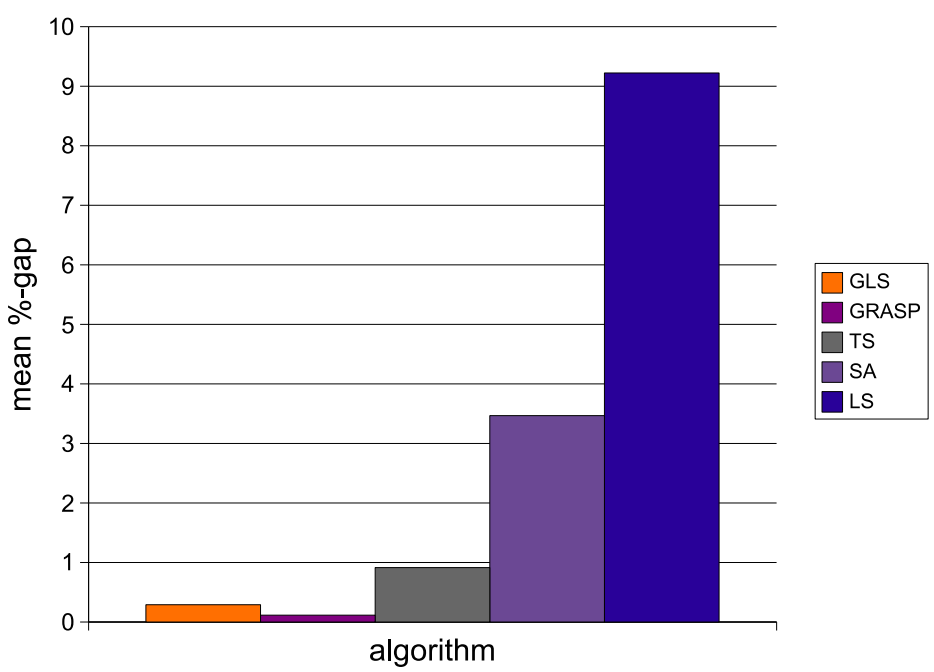


Figure 6.1: Overall mean %-gap

has more than one. This capability is allegable with the major strengths of these algorithms. GLS gradually moves away from attractive solutions and therefor the embedded random local search is able to reach widespread areas of the search space. GRASP operates somewhat different, its strenghts lies the randomized greed heuristics which, in the optimal case, produces good starting solutions for the embedded local search procedure, which are distributed among the whole search space.

An other important feature that table 6.11 and 6.12 show is that the quality of the obtained solutions only decreases somewhat and so these solutions, altough not global optimal, can be adequate, too.

Tabu search also achieved good results in terms of the mean %-gap. However, it has reached significant fewer global optima than guided local search or GRASP and the deviation values indicate that the stability of the search process is somewhat deteriorated. A reason for this behavior is the fixed tabulist length which could cause a either a lockout of interesting regions of the search space when the tabulist is too long or the search process gets stuck around a local optimum when the tabulist is too short.

The results obtained with the simulated annealing metaheuristc were mixed. For some problem instances, especially instances from the bur collection and smaller instances from the other collections, simulated annealing clearly performs better than tabu search. On the other side some results, e.g. for instances from the chr collection, are not very satisfactory. This indicates that simulated annealing, in its

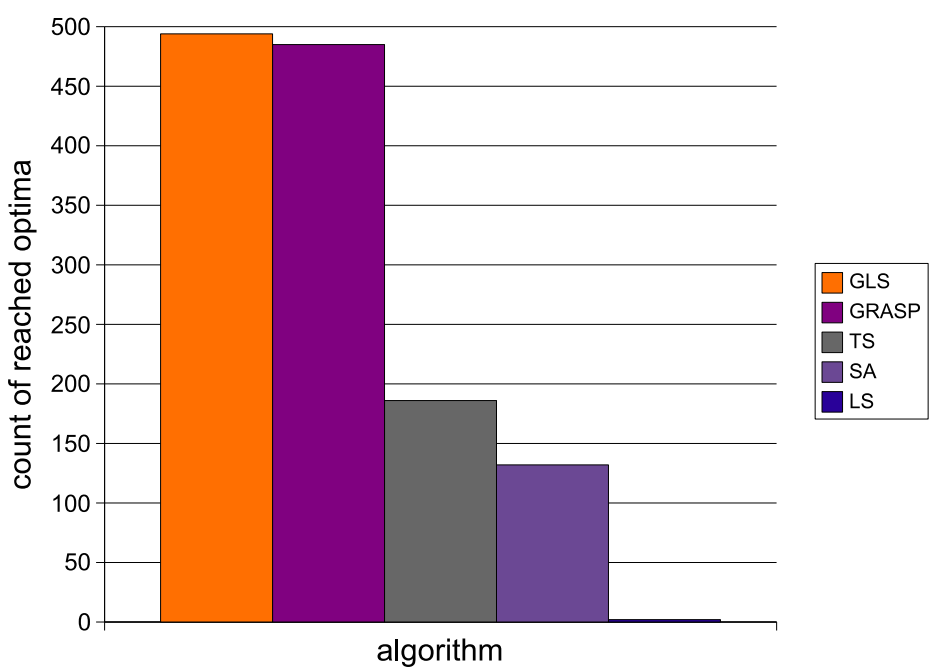


Figure 6.2: Overall count of reached global optima

traditional fashion, suffers from a worse stability of the obtained results when applied to the quadratic assignment problem, which could be an effect of the geometric cooling schedule. An improvement like reheating or an occasional perturbation phase might yield better solutions.

As expected local search only yields a few global optimal solutions but although no global optimal solution was found for many problem instances especially from the bur set the obtained solutions were nearly optimal. This implies that the chosen neighborhood structure is well suited for solving the quadratic assignment problem which is certainly important for the other metaheuristics, too. However, it is somewhat surprising that only results for the instances from the chr set were very unsatisfactory.

The following tables show the results of our tests per algorithm and test instance.

	Optimum		%-gap		
Instance	absolute	count	Best	Mean	Deviation
bur26a	5426670	0	0.13	0.35	0.17
bur26b	3817852	0	0.20	0.50	0.22
bur26c	5426795	0	0.00	0.36	0.36
bur26d	3821225	0	0.02	0.46	0.49
bur26e	5386879	0	0.01	0.36	0.31
bur26f	3782044	0	0.02	0.43	0.37
bur26g	10117172	0	0.02	0.38	0.32
bur26h	7098658	0	0.02	0.43	0.34
chr12a	9552	1	0.00	44.35	31.81
chr15a	9896	0	23.54	49.39	22.19
chr20a	2192	0	18.80	47.13	15.03
nug12	578	0	2.08	5.81	2.84
nug14	1014	0	1.18	4.52	1.58
nug15	1150	0	1.04	4.75	2.10
nug20	2570	0	2.33	4.65	1.34
nug25	3744	0	1.12	3.92	1.69
nug30	6124	0	2.06	4.46	1.07
tai10a	135028	0	0.59	5.65	3.12
tai12a	224416	1	0.00	8.82	3.37
tai15a	388214	0	1.86	4.61	1.78
tai17a	491812	0	2.74	5.89	1.67
tai20a	703482	0	2.41	5.67	1.62

Table 6.8: Local Search results

	Optimum		%-gap		
Instance	absolute	count	Best	Mean	Deviation
bur26a	5426670	1	0.00	0.14	0.05
bur26b	3817852	7	0.00	0.14	0.09
bur26c	5426795	2	0.00	0.03	0.05
bur26d	3821225	1	0.00	0.03	0.08
bur26e	5386879	6	0.00	0.01	0.01
bur26f	3782044	1	0.00	0.09	0.14
bur26g	10117172	2	0.00	0.02	0.01
bur26h	7098658	2	0.00	0.06	0.17
chr12a	9552	18	0.00	4.18	7.74
chr15a	9896	2	0.00	16.68	10.72
chr20a	2192	0	7.21	33.85	50.55
nug12	578	23	0.00	0.61	2.77
nug14	1014	5	0.00	3.66	5.53
nug15	1150	8	0.00	1.84	5.06
nug20	2570	3	0.00	2.44	5.05
nug25	3744	4	0.00	2.93	6.88
nug30	6124	1	0.00	2.58	5.77
tai10a	135028	23	0.00	0.16	0.56
tai12a	224416	20	0.00	0.62	1.34
tai15a	388214	1	0.00	1.58	0.93
tai17a	491812	1	0.00	1.75	1.00
tai20a	703482	1	0.00	2.89	2.83

Table 6.9: Simulated Annealing results

	Optimum		%-gap		
Instance	absolute	count	Best	Mean	Deviation
bur26a	5426670	3	0.00	0.19	0.11
bur26b	3817852	0	0.02	0.37	0.19
bur26c	5426795	4	0.00	0.22	0.28
bur26d	3821225	0	0.00	0.31	0.37
bur26e	5386879	2	0.00	0.25	0.27
bur26f	3782044	1	0.00	0.40	0.42
bur26g	10117172	2	0.00	0.13	0.17
bur26h	7098658	0	0.00	0.45	0.34
chr12a	9552	20	0.00	1.18	2.41
chr15a	9896	25	0.00	0.00	0.00
chr20a	2192	1	0.00	8.85	6.46
nug12	578	8	0.00	1.36	1.40
nug14	1014	16	0.00	0.87	1.55
nug15	1150	7	0.00	1.26	1.56
nug20	2570	5	0.00	1.16	1.10
nug25	3744	9	0.00	0.63	1.14
nug30	6124	4	0.00	0.75	1.09
tai10a	135028	23	0.00	0.04	0.13
tai12a	224416	25	0.00	0.00	0.00
tai15a	388214	19	0.00	0.19	0.87
tai17a	491812	5	0.00	1.00	0.95
tai20a	703482	7	0.00	0.52	0.81

Table 6.10: Tabu Search results

	Optimum		%-gap		
Instance	absolute	count	Best	Mean	Deviation
bur26a	5426670	25	0.00	0.00	0.00
bur26b	3817852	25	0.00	0.00	0.00
bur26c	5426795	25	0.00	0.00	0.00
bur26d	3821225	24	0.00	0.00	0.00
bur26e	5386879	25	0.00	0.00	0.00
bur26f	3782044	25	0.00	0.00	0.00
bur26g	10117172	25	0.00	0.00	0.00
bur26h	7098658	25	0.00	0.00	0.00
chr12a	9552	25	0.00	0.00	0.00
chr15a	9896	10	0.00	0.93	0.85
chr20a	2192	1	0.00	5.22	3.47
nug12	578	25	0.00	0.00	0.00
nug14	1014	25	0.00	0.00	0.00
nug15	1150	25	0.00	0.00	0.00
nug20	2570	25	0.00	0.00	0.00
nug25	3744	25	0.00	0.00	0.00
nug30	6124	25	0.00	0.00	0.00
tai10a	135028	25	0.00	0.00	0.00
tai12a	224416	25	0.00	0.00	0.00
tai15a	388214	25	0.00	0.00	0.00
tai17a	491812	25	0.00	0.00	0.00
tai20a	703482	9	0.00	0.26	0.22

Table 6.11: Guided Local Search results

	Optimum		%-gap		
Instance	absolute	count	Best	Mean	Deviation
bur26a	5426670	25	0.00	0.00	0.00
bur26b	3817852	25	0.00	0.00	0.00
bur26c	5426795	25	0.00	0.00	0.00
bur26d	3821225	25	0.00	0.00	0.00
bur26e	5386879	25	0.00	0.00	0.00
bur26f	3782044	25	0.00	0.00	0.00
bur26g	10117172	25	0.00	0.00	0.00
bur26h	7098658	25	0.00	0.00	0.00
chr12a	9552	25	0.00	0.00	0.00
chr15a	9896	25	0.00	0.00	0.00
chr20a	2192	0	0.18	2.02	1.56
nug12	578	25	0.00	0.00	0.00
nug14	1014	25	0.00	0.00	0.00
nug15	1150	25	0.00	0.00	0.00
nug20	2570	25	0.00	0.00	0.00
nug25	3744	24	0.00	0.00	0.01
nug30	6124	1	0.00	0.30	0.16
tai10a	135028	25	0.00	0.00	0.00
tai12a	224416	25	0.00	0.00	0.00
tai15a	388214	25	0.00	0.00	0.00
tai17a	491812	25	0.00	0.00	0.00
tai20a	703482	10	0.00	0.23	0.21

Table 6.12: GRASP results

In the following figures the results of the algorithms are grouped per test instance to show which instances were tackled best by what algorithms.

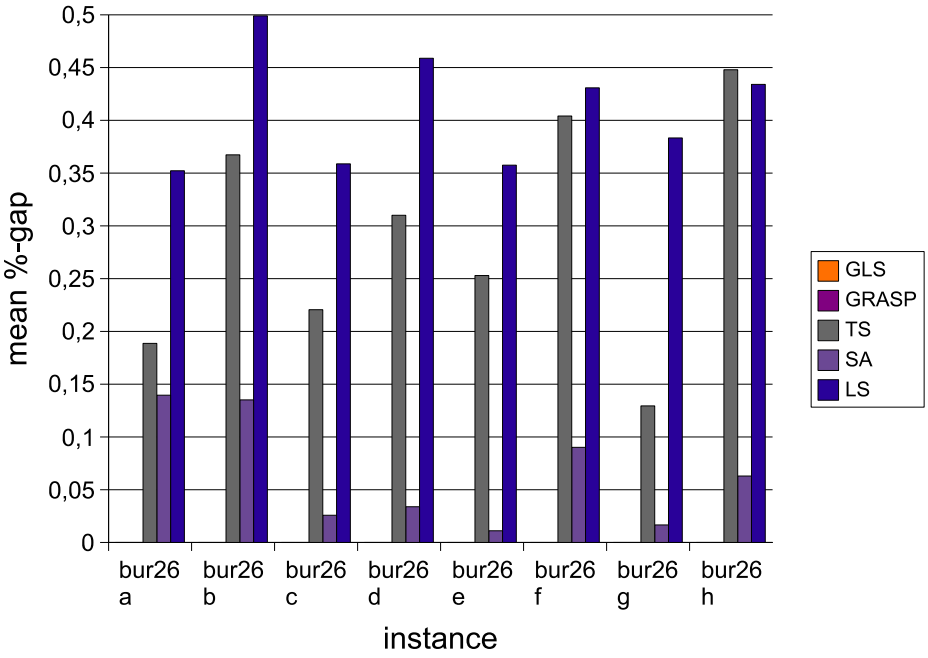


Figure 6.3: Mean %-gap for bur Instances

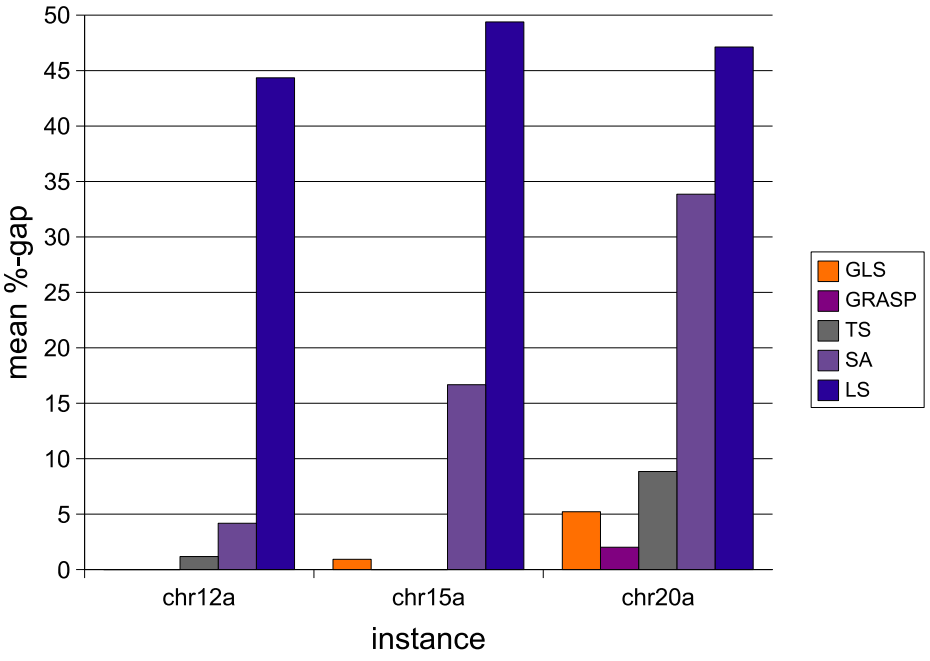


Figure 6.4: Mean %-gap for chr instances

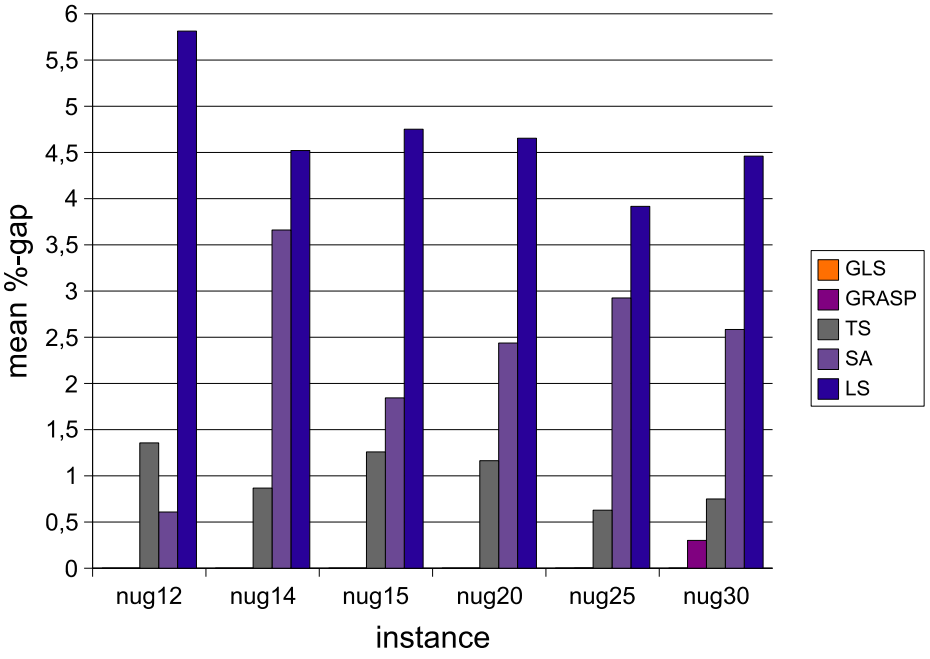


Figure 6.5: Mean %-gap for nug instances

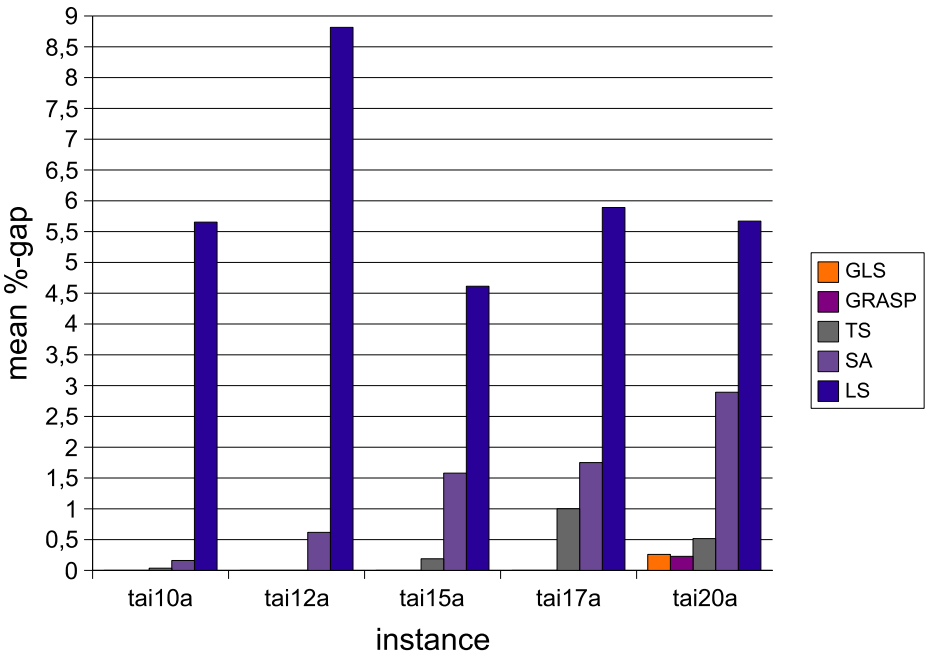


Figure 6.6: Mean %-gap for tai instances

A conclusion is the place where you got tired of thinking

Steven Wright

Chapter 7

Conclusions

After an elaborated introduction of the quadratic assignment problem and the implemented metaheuristics this thesis presented a generic library for metaheuristics and its application to the QAP.

The already existing foundations of the EAlib library have been improved to meet the requirements for the new generic metaheuristics. In particular interfaces classes have been introduced that allow fine grained modelling of new classes and support runtime queries for implemented features of specific components. Additionally the parameter handling mechanism has been extended with parameter groups that allow to denote different groups of parameter values for different components in EAlib.

With this enhanced EAlib generic versions of the local search, simulated annealing, tabu search, guided local search and greedy randomized adapetive search procedure metaheuristics have been implemented. The latter two algorithms also introduced an efficient way of handling an embedded algorithm.

All considered metaheuristics have then been applied to the quadratic assignment problem which showed that only some special parts need to be implemeted separately and most of the problem dependent sourcecode can be shared among all algorithms involved.

Of course there are many interesting and useful ideas and task left open for future work.

- implement more interesting algorithms like antcolony optimization, variable neighborhood search or iterated local search,
- add more “standard” features to the implemented algorithms, e.g. dynamic tabulist length, reheating, disturbance methods
- itegrate useful template chromosomes, e.g. a generic string chromosome,
- provide additional language bindings e.g. for Java and C#.

List of Algorithms

List of Figures

List of Tables

Bibliography

[1] Anstreicher, K. M., and Brixius, N. W. A New Bound for the Quadratic Assignment Problem Based on Convex Quadratic Programming. *Mathematical Programming 89*, 3 (2001), 341–357.

[2] Anstreicher, K. M., Brixius, N. W., Goux, J.-P., and Linderoth, J. Solving large quadratic assignment problems on computational grids. *Mathematical Programming 91*, 3 (February 2002), 563–588.

[3] Battiti, R., and Tecchiolli, G. The reactive tabu search. *ORSA Journal on Computing 6*, 2 (1994), 126–140.

[4] Blum, C., and Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys 35*, 3 (2003), 268–308.

[5] Brixius, N. W., and Anstreicher, K. M. Solving quadratic assignment problems using convex quadratic programming relaxations. *Optimization Methods and Software 16* (2001), 49–68.

[6] Brixius, N. W., and Anstreicher, K. M. The Steinberg Wiring Problem. In *The Sharpest Cut: The Impact of Manfred Padberg and His Work*, M. Grötschel, Ed. SIAM, June 2004.

[7] Burkard, R., Karisch, S., and Rendl, F. QAPLIB - A Quadratic Assignment Problem Library. *Journal of Global Optimization 10* (1997), 391–403. http://www.seas.upenn.edu/qaplib/.

[8] Cerny, V. Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *Journal of Optimization Theory and Applications 45*, 1 (1985), 41–51.

[9] Commander, C. W., and Pardalos, P. M. A Survey of the Quadratic Assignment Problem, with Applications. Submitted to The Moreahead Electronic Journal of Applicable Mathematics, April 2003.

[10] Edwards, C. The derivation of a greedy approximator for the Koopmans–Beckmann quadratic assigment problem. In *Proceedings of the 77-th Combinatorial Programming Conference (CP77)* (1977), pp. 55–86.

[11] Feo, T. A., and Resende, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters 8* (April 1989), 67–71.

[12] Feo, T. A., and Resende, M. G. C. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization 6*, 2 (March 1995), 109–133.

[13] Frazer, M. Exact Solution of the Quadratic Assignment Problem, April 1997.

[14] Garey, M. R., and Johnson, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. A series of books in the mathematical sciences. W.H. Freeman and Company, New York, NY, 1979.

[15] Gilmore, P. C. Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem. *SIAM Journal on Applied Mathematics 10* (1962), 305–313.

[16] Glover, F. W. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research 13*, 5 (May 1986), 533–549.

[17] Glover, F. W. Tabu Search — Part I. *ORSA Journal on Computing 1*, 3 (1989), 190–206.

[18] Glover, F. W. Tabu Search — Part II. *ORSA Journal on Computing 2*, 1 (1990), 4–32.

[19] Glover, F. W., and Kochenberger, G. A., Eds. *Handbook of Metaheuristics*, vol. 57 of *International series in operations research and management science*. Kluwer Academic Publishers, Boston Hardbound, 2003.

[20] Gutin, G., and Yeo, A. Polynomial approximation algorithms for the TSP and the QAP with a factorial domination number. *Discrete Applied Mathematics 119*, 1–2 (June 2002), 107–116.

[21] Henderson, D., and Jacobson, S. H. The Theory and Practice of Simulated Annealing. In *Handbook of Metaheuristics*, F. W. Glover and G. A. Kochenberger, Eds., vol. 57 of *International series in operations research and management science*. Kluwer Academic Publishers, Boston Hardbound, 2003, ch. 10, pp. 287–319.

[22] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by Simulated Annealing. *Science 220*, 4598 (May 1983), 671–680.

[23] Koopmans, T. C., and Beckmann, M. Assignment Problems and the Location of Economic Activities. *Econometrica, Journal of the Econometric Society 25*, 1 (January 1957), 53–76.

[24] Kuhn, H. W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly 2* (1955), 83–97.

[25] Lawler, E. L. The Quadratic Assignment Problem. *Management Science 9* (1963), 586–599.

[26] Li, Y., Pardalos, P. M., and Resende, M. A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem. In *Quadratic assignment and related problems*, P. M. Pardalos and H. Wolkowicz, Eds., vol. 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1994, pp. 237–261.

[27] Loiola, E. M., de Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., and Querido, T. An analytical Survey for the Quadratic Assignment Problem, 2004.

[28] Lourenço, H. R., Martin, O. C., and Stützle, T. A Beginner's Introduction to Iterated Local Search. In *Proceedings of MIC'2001—Meta–heuristics International Conference* (July 2001), vol. 1, pp. 1–6. Porto, Portugal.

[29] Lourenço, H. R., Martin, O. C., and Stützle, T. Iterated Local Search. In *Handbook of Metaheuristics*, F. W. Glover and G. A. Kochenberger, Eds., vol. 57 of *International series in operations research and management science*. Kluwer Academic Publishers, Boston Hardbound, 2003, ch. 11, pp. 321–353.

[30] Martí, R. Multi-Start Methods. In *Handbook of Metaheuristics*, F. W. Glover and G. A. Kochenberger, Eds., vol. 57 of *International series in operations research and management science*. Kluwer Academic Publishers, Boston Hardbound, 2003, ch. 12, pp. 355–368.

[31] Metropolis, N., Rosenbluth, A., Rosenbluth, M. N., Teller, A. H., and Teller, E. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics 21*, 6 (June 1953), 1088–1092.

[32] Mills, P., Tsang, E. P. K., and Ford, J. Applying an extended Guided Local Search to the Quadratic Assignment Problem. *Annals of Operations Research 118*, 1 (Feburary 2003), 121–135.

[33] Padula, S. L., and Kincaid, R. K. Aerospace Applications of Integer and Combinatorial Optimization. NASA Technical Memorandum 110210, NASA, Langley Research Center, Hampton, Virginia 23681-0001, October 1995.

[34] Palubeckis, G. An Algorithm for Construction of Test Cases for the Quadratic Assigmnet Problem. *Informatica 11*, 3 (2000), 281–296.

[35] Raidl, G. EAlib 1.1 – A Generic Library for Metaheuristics. Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2004.

[36] Rendl, F., and Sotirov, R. Bounds for the Quadratic Assignment Problem Using the Bundle Method, August 2003.

[37] Resende, M. G. C. Greedy Randomized Adaptive Search Procedures. In *Handbook of Metaheuristics*, F. W. Glover and G. A. Kochenberger, Eds., vol. 57 of *International series in operations research and management science*. Kluwer Academic Publishers, Boston Hardbound, 2003, ch. 8, pp. 219–249.

[38] Resende, M. G. C., and Ribeiro, C. C. Parallel Greedy Randomized Adaptive Search Procedures. Tech. Rep. TD-67EKXH, AT&T Labs Research, December 2004.

[39] Sahni, S., and Gonzales, T. P-Complete Approximation Problems. *Journal of the ACM 23*, 3 (July 1976), 555–565.

[40] Steinberg, L. The Backboard Wiring Problem: A Placement Algorithm. *SIAM Review 3*, 1 (1961), 37–50.

[41] Taillard, E. D. Robust taboo search for the quadratic assignment problem. *Parallel Computing 17*, 4–5 (July 1991), 443–455.

[42] Tsang, E. P. K., and Voudouris, C. Fast Local Search and Guided Local Search and Their Application to British Telecom's Workforce Scheduling Problem. Tech. Rep. CSM-246, Department of Computer Science, University of Essex, Colchester CO4 3SQ, August 1995.

[43] Tsang, E. P. K., and Wang, C. J. A Generic Neural Network Approach for Constraint Satisfaction Problems. In *Neural Network Applications*, J. G. Taylor, Ed. Springer-Verlag, 1992, pp. 12–22.

[44] Voudouris, C., and Tsang, E. P. K. Guided Local Search. Tech. Rep. CSM-247, Department of Computer Science, University of Essex, Colchester, C04 3SQ, UK, August 1995.

[45] Voudouris, C., and Tsang, E. P. K. Guided Local Search. In *Handbook of Metaheuristics*, F. W. Glover and G. A. Kochenberger, Eds., vol. 57 of *International series in operations research and management science*. Kluwer Academic Publishers, Boston Hardbound, 2003, ch. 7, pp. 185–217.