- Diplomarbeit -

(Implementierungsarbeit)

Adaptive Ant Colony Optimization for the Traveling Salesman Problem

Michael Maur

Mat.-Nr.: 1192603

<lastname>@stud.tu-darmstadt.de

Eingereicht im Dezember 2009

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Rechts- und Wirtschaftswissenschaften

Fachgebiet Operations Research

Prof. Dr. Wolfgang Domschke

Betreuer: Dr. Thomas Stützle (IRIDIA, ULB)

Table of Contents

List of Figures

List of Tables

1 Introduction

The present work deals with approaches to control the parameters of Ant Colony Optimization (ACO) algorithms for the Traveling Salesman Problem (TSP). The TSP is a combinatorial optimization problem. It is concerned with a salesman that needs to plan a tour on which he visits a given set of n cities before finally returning to his starting point. The goal is to find the order in which he should visit the cities so that the overall length of the tour is minimized. Many practical applications of the TSP exist, such as tour planning for packet delivery services.

Many popular methods to solve the TSP model it as a graph problem. The nodes in the graph represent the cities, which the salesman needs to visit and the edges correspond to the available connections. The edges are assigned a weight d_{ij} that corresponds to the distance between the two neighboring cities i and j. To simplify the model, the graph is fully connected and the salesman is obliged to visit every city exactly once. As the present work deals with the symmetric TSP, the edges in the graph are undirected, i.e., the distance that needs to be traveled to get from city i to city j is the same as in the opposite direction. In terms of graph theory, finding the salesman's optimal tour corresponds to finding the shortest Hamilton cycle in the graph.

The TSP is of particular interest in an optimization context because it is a NP-hard problem, i.e., there is no optimization technique that generally solves it in polynomial time [19]. Therefore, large TSP instances are typically solved using heuristic approaches, i.e., methods that derive rather good solutions within a relatively short amount of time, but which do not guarantee the optimality of the returned result.

This is where Ant Colony Optimization comes into play. ACO algorithms are heuristic methods that mimic the foraging behavior of real ants. When a real ant finds a food source, it will determine the desirability of using this food source, which may be impacted by the amount of food that was found or its distance to the nest. While traveling back to the nest, the ant marks its way with a chemical known as pheromone. For some real ant species, the amount of deposited pheromone depends on the desirability of using the food source.

Whenever another ant explores the region around the nest on its search for food, the trail it chooses will be biased towards directions in which it senses a lot of pheromone on the ground. Over time, this eventually leads to more and more ants using the promising paths and depositing additional pheromone on them. Should the reserves at a popular food source get depleted over time, the natural evaporation of the chemical gives the pheromone trails the chance to adapt to the new situation.

It should be noted that the ants do not communicate directly. The only way they exchange information is by depositing pheromone on promising trails. This type of communication by means of physical modifications of the environment is called stigmergy.

Using the previously introduced graph representation of the TSP, the same principle can be used to create a computer algorithm in which m artificial ants evolve possible choices for the salesman's tour. Instead of looking for food in a natural environment, the artificial ants travel the graph, trying to construct tours of minimal length that pass by each node exactly once. As an equivalent to the real ants' pheromone trails, each edge in the graph stores a value reflecting the amount of artificial pheromone that has been deposited on it.

The artificial ants' tour construction process can be depicted as a sequence of decisions, which city to choose next. A particular unvisited city's chance of being selected depends on three factors: a) the amount of pheromone on the edge leading to that city, b) a heuristic value reflecting the length of this edge, and c) a certain degree of random exploration. The use of the heuristic value allows the ant to take into account her potential choices' immediate cost. After all ants returned to the node from which they started, artificial pheromone is registered on the edges used by the ants. The shorter the tour constructed by an ant, the larger is the amount of pheromone that is added on the edges it used in its tour. To simulate the evaporation of the pheromone, all edges' pheromone levels are decreased by a certain percentage before the pheromone deposit takes place.

This process (i.e., tour construction and pheromone update) is consecutively repeated until the algorithm run is eventually terminated. Certain edges that are frequently encountered as part of short tours will obtain high pheromone levels, which increases their chance of being selected in subsequent iterations.

An ACO algorithm's strength in solving optimization problems like the TSP is fundamentally based on its ability to balance two factors. On the one hand, the algorithm needs to be able to focus on promising regions of the search space by exploiting the knowledge that gets evolved in the form of the edges' different pheromone levels. On the other hand, a sufficient exploration of the search space needs to be ensured by introducing the right amount of diversity into the tour construction process. In order to establish and sustain this balance, choosing a good parameter configuration is crucial.

Typically, a fair amount of research is done on determining good static parameter settings. However, the obtained configurations are usually dependent on the algorithm's runtime, at the end of which the achieved solution quality is evaluated. Measuring the performance after a different period of time would oftentimes lead to the identification of other parameter settings that are better suited to that situation. This indicates that different parameter settings may be optimal during different stages of the run.

One could argue further that in the case of non-deterministic search methods, such as ACO, taking only static criteria into account may be insufficient. A parameter configuration's performance in a specific situation may also depend on factors inherent in the search processes' state at that particular point in time. E.g., when observing a stagnation of the search (i.e., most ants keep constructing the same tours), changing parameters to intensify the exploration of the search space would be an appropriate choice to improve algorithm performance. However, doing the same in a situation without stagnation may oftentimes impair the algorithm's performance.

Motivated by these needs, it is the main concern of this thesis to investigate possible options to set ACO parameters in a way that ensures a reasonably good performance for a rather large bandwidth of different runtimes.

The remainder of this document is structured as follows. Chapter 2 provides the reader with a more detailed insight into ACO algorithms. Four popular variants are illustrated with respect to the functionality they share and what their differences are. The third chapter starts by reviewing a classification scheme for parameter setting methods. It then uses the introduced terminology to present a survey of related work in the fields of Evolutionary Algorithms and Ant Colony Optimization. Details on the experimental setup used in three subsequently presented experimental studies are provided in chapter 4. The first study, which is presented in chapter 5, investigates the trade-off between different fixed parameter settings. Chapter 6 presents results for a series of methods that use predefined schedules to change an ACO algorithm's parameters while it is executed. Finally, chapter 7 introduces and evaluates an adaptive ACO variant, i.e., a method changing an algorithm parameter based on how certain characteristics of the search process develop. The work is concluded in chapter 8, which also gives some suggestions on how it could be continued in the future.

2 Ant Colony Optimization

The following sections will introduce the reader to four Ant Colony Optimization algorithms, which researchers examined in a TSP context. Ant system, as explained in section 2.1, has been chosen primarily for historical reasons. It was the first ACO approach to be published and served as the basis for the other methods illustrated in this chapter. Sections 2.2 through 2.4 introduce Ant Colony System, MAX-MIN Ant System, and Rank-based Ant System, which are essentially extensions of Ant System. Even though Rank-based Ant System did not receive as much attention in literature as the previous two approaches, the underlying concept appeared promising and performance gains were expected from optimizing its parameters.

In parts, this chapter draws from chapter 3 of the book on ACO published by Dorigo and Stützle [22] in 2004, which the interested reader may use as a source for an in-depth study of the topic.

2.1 Ant System

As previously mentioned, Ant System (AS) was the first ACO approach to be published and various extensions of it exist. For an overview of the latter, the reader is referred to the book by Dorigo and Stützle. What today is commonly identified as Ant System corresponds to one out of three algorithm variants proposed by Dorigo et al. [20] in 1991. For performance reasons, the research community focused on the variant introduced as *ant-cycle* and abandoned *ant-density* and *ant-quantity*.

The general structure of any ACO algorithm, and thus also AS, is relatively simple. Starting with an initialization of the algorithm, iteration after iteration all ants first construct their tour and then update the pheromone trails accordingly. In an extended scenario (i.e., an optional case), a local search method can be used to improve the ants' tours before updating the pheromone. This idea is summarized in the following pseudo code (a similar representation is used in [22]):

```
initialization();
while (not(terminationCriterionTrue())) {
  constructAntTours();
  if (useOptionalLocalSearch)
    applyLocalSearch();
  updatePheromones();
}
```

Subsequently, these steps are examined in more detail. The list below gives an overview of the variables and parameters that will be used. This overview as well as subsequent ones will show user-provided parameters separate from those that are introduced primarily to illustrate the algorithm's functionality. Variables will be shown before sets, both in alphabetical order. A complete list of all used designators can be found on page 77, i.e., at the end of this thesis.

α	relative influence of the pheromone trail during tour construction
β	relative influence of the heuristic values during tour construction
m	number of ants
n	number of cities
ρ	pheromone evaporation level for the global pheromone update

C^k	cost, resp. length of the tour built by ant k
C^{nn}	cost, resp. length of a nearest neighbor tour
d_{ij}	distance from city i to city j
η_{ij}	heuristic value of the edge between city i and city j
${}_{rp}p_{ij}^k$	probability with which ant k, located at city i, continues its tour to city j according to the random proportional rule
τ_0	initial pheromone level
τ_{ij}	pheromone on edge between cities i and j
τ_{ij}^p	value of τ_{ij} prior to applying a specific update
$\Delta\tau_{ij}^k$	amount of pheromone deposited by ant k on the edge between cities i and j
M	set of all ants
N	set of all cities
N_i^k	set of cities that ant k has not yet visited when being at city i
T^k	tour generated by ant k (i.e., an ordered set of edges)

The remainder of this section puts the above into context while illustrating the algorithm's functionality. What follows will be structured according to the ACO pseudo code shown at the beginning of this section.

Initialization: An ACO algorithm's initialization phase is generally concerned with getting values assigned to all parameters and with setting up the pheromone trails (i.e., all edges between cities are assigned an initial amount of pheromone). Formula 2.1 determines the initial pheromone level τ_0 as used in AS:

$$\tau_0 = \frac{m}{C^{nn}} \tag{2.1}$$

The intuition behind the formula is to have roughly as much pheromone on all edges as would typically be deposited in one iteration, if all ants had used the specific edge. The value for τ_0 comes about as there are m ants (i.e., a parameter specified by the user) which each deposit an amount of pheromone that is inversely related to the length of their tour. The value C^{nn} is a crude approximation of the possible range of tour lengths generated by the ants. It is determined using the *nearest neighbor heuristic* [45] starting from a randomly chosen city.

It is important to start from a reasonably sized pheromone level τ_0. Too large values would require a lot of pheromone to be evaporated before the ants' pheromone traces are able to impact the search in a relevant manner. For too small values however, the tours generated in the first iterations would impact the search too strongly. The latter would possibly enforce the detailed exploration of suboptimal regions of the search space.

Somewhat different formulas for τ_0 exist for Ant Colony System, MAX-MIN Ant System and Rank-based Ant System and will be introduced in the respective sections.

Equation 2.2 derives the heuristic value η_{ij} for all edges. The latter is inversely related to the respective distance. I.e., the smaller the distance, the higher is the heuristic desirability to choose an edge.

$$\eta_{ij} = \frac{1}{d_{ij} + 0.1} \qquad \forall i, j \in N \tag{2.2}$$

In the present implementation, the distance is increased by a small constant to account for the possible case of a zero distance (i.e., to avoid a division by zero). Zero distances can be encountered as the algorithm uses the Euclidean distance between cities, rounded to the nearest whole number. The value of 0.1 can be considered appropriate in the given context as typical distances in the used problem instances are larger by several orders of magnitude. Thus, the heuristic information is not significantly falsified by this adaptation. The reader should also note that the precision of the heuristic information does not have any impact on the algorithm's accuracy in measuring a tour's length, i.e., on its ability to assess the quality of a solution.

Tour construction: The first step to be repeated in every iteration of an ACO algorithm is the tour construction. Generally, ant algorithms work with a population of m ants, which each construct one possible solution per algorithm iteration. Starting at a random city, an ant constructs a tour (i.e., a solution to the TSP), visiting each city exactly once and finally returning to the one where it started. At each city, the next city is chosen according to the *random proportional rule*. By means of equations 2.3 and 2.4, the ant determines the probability of moving from its present location, which is identified by index i, to the city with index j.

$$_{rp}p_{ij}^{k} = \frac{(\tau_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum_{l \in N_i^k} (\tau_{il})^{\alpha} \cdot (\eta_{il})^{\beta}} \qquad \forall k \in M, i \in N, j \in N_i^k \tag{2.3}$$

$$_{rp}p_{ij}^{k} = 0 \qquad \forall k \in M, i \in N, j \notin N_i^k \tag{2.4}$$

N_i^k denotes the *feasible neighborhood* of ant k when located at city i, i.e., the cities the ant has not yet visited on its tour. The probability to choose a previously unvisited city is based on the term $(\tau_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}$. The latter combines the pheromone located on an edge with its heuristic value in a multiplicative manner. Accordingly, in the general case ($\alpha, \beta > 0$), the probability of choosing a specific edge increases with both, the size of its heuristic value and the amount of pheromone deposited on it. The ratio of α and β controls the relative importance of pheromone and heuristic information. Even though there was some interest in the past in finding optimal values for both parameters, most present work keeps α stable at 1.0 while choosing a β larger than 1.0.

After all ants constructed their tours and before updating the pheromones, there is the possibility to apply a local search method to the tours built by the ants. Generally, all such methods try to improve a given solution (i.e., a tour built by an ant) by examining the solution space close to it. If a local search method is applied, the algorithm uses the improved solutions it generates during the subsequently described pheromone update. Without local search, the update is performed with the original tours as constructed by the ants. More information on the respective method used in the present work and the circumstances under which it is applied are provided in subsection 4.2.3.

Pheromone update: As the next step in standard Ant System, all ants update the pheromone on the tours they built. Equation 2.5 depicts the *pheromone update rule.*

$$\tau_{ij} = (1-\rho) \cdot \tau_{ij}^{p} + \sum_{k \in M} \Delta\tau_{ij}^{k} \qquad \forall i,j \in N \tag{2.5}$$

It reflects the evaporation of previously deposited pheromone as well as the deposit of new pheromone by the ants. The pheromone level prior to the update, herein denoted as τ_{ij}^{p}, is in a first step reduced by a fixed percentage determined by the algorithm parameter ρ. Subsequently, the pheromone on an edge is increased by a value $\Delta\tau_{ij}^{k}$ for each ant k that used it in its tour. Equations 2.6 and 2.7 define the amount by which the pheromone values increase.

$$\Delta\tau_{ij}^{k} = 1/C^{k} \qquad \forall (i,j) \in T^{k}, k \in M \tag{2.6}$$

$$\Delta\tau_{ij}^{k} = 0 \qquad \forall (i,j) \notin T^{k}, k \in M \tag{2.7}$$

Each ant increases the edges it travels by an amount inversely proportional to the length of its tour; i.e., ants with long tours deposit less pheromone than ants with shorter tours. An ant k's tour T^{k} is a set of tuples (i,j) describing the end points of the different edges composing the tour, ordered in the way they were visited by the ant. The intuitive notion of a tour's length is explicitly defined by equation 2.8 as the sum of the traveled edges' length.

$$C^{k} = \sum_{(i,j) \in T^{k}} d_{ij} \qquad \forall k \in M \tag{2.8}$$

2.2 Ant Colony System

Ant Colony System (ACS) was proposed by Dorigo and Gambardella [21] in 1997 as an extension of AS. Even though the two algorithms have much in common, all major phases (i.e., initialization, tour construction, and pheromone update) bear differences.

Initialization: For ACS, the choice of the initial pheromone level (i.e., τ_0) is a different one. Equation 2.1 is replaced by equation 2.9, thereby lowering τ_0 by a factor of $(m \cdot n)$.

$$\tau_0 = \frac{1}{n \cdot C^{nn}} \tag{2.9}$$

Tour construction: The ACS tour construction process comes with a series of innovations compared to AS. In this context, three new parameters and two new variables will be used. Again, the former can be changed by the algorithm user, while the latter are utilized solely in explaining the algorithm's functionality.

cand	number of cities on the candidate list
q_0	probability of simply choosing the *best* city according to the random proportional rule instead of applying the rule as described for AS
ξ	pheromone evaporation level for the local pheromone update

bs	alias for the best-so-far ant, resp. tour
${}_{pp}p_{ij}^k$	probability with which ant k, located at city i, continues its tour to city j according to the pseudorandom proportional rule

The first innovation is concerned with the way the ants choose the next city to go to. Ant System uses the random proportional rule (i.e., equation 2.3) to determine a probability of selection for each unvisited neighboring city. The ACS approach is based on these values, but strongly boosts the use of the city having the highest probability ${}_{rp}p_{ij}^k$. A parameter q_0 determines the probability of choosing this *best* city instead of distributing the chances proportional to the ${}_{rp}p_{il}^k$. The latter is done only with the remaining probability of $(1-q_0)$. This new behavior has been labeled *pseudorandom proportional* selection of the next city. Its effect on the respective selection probabilities is captured in the following equations:

$${}_{pp}p_{ij}^k = q_0 + (1-q_0) \cdot {}_{rp}p_{ij}^k \qquad \forall i,j \in N, k \in M : j = argmax_{l \in N_i^k} \left\{ {}_{rp}p_{il}^k \right\} \tag{2.10}$$

$${}_{pp}p_{ij}^k = (1-q_0) \cdot {}_{rp}p_{ij}^k \qquad \forall i,j \in N, k \in M : j \neq argmax_{l \in N_i^k} \left\{ {}_{rp}p_{il}^k \right\} \tag{2.11}$$

It needs to be noted that in contrast to AS, ACS uses a fixed value of α equal to 1.0 in the random proportional rule.

The second innovation with respect to tour construction deals with pheromone evaporation. While AS uses evaporation only as part of the so-called *global pheromone trail update* (i.e., after all tours were constructed), ACS also evaporates pheromone during the construction phase. Generally, as an ant moves from city to city, it removes some pheromone from the edges it uses. This behavior encourages a more exploring behavior of the algorithm by making previously chosen edges less desirable. This effectively prevents a stagnation of the search [21]. The new parameter $\xi \in [0,1]$ determines the strength of this new type of evaporation, which has been labeled *local pheromone trail update*:

$$\tau_{ij} = (1-\xi) \cdot \tau_{ij}^p + \xi \cdot \tau_0 \qquad \forall (i,j) \in T^k, k \in M \tag{2.12}$$

This rule is applied each time an ant k takes a construction step, the variables i and j referring to the endpoints of the edge it just added. The term $\xi \cdot \tau_0$ ensures that the amount of pheromone on all used trails approaches its initialization level, i.e., that it does not converge towards 0. The designator τ_{ij}^{p} again refers to the value of τ_{ij} before the respective update.

A third difference to AS tour construction is the explicit use of a parallel tour construction method. In AS, it does not matter whether the tours get constructed in parallel or one after the other, because tour construction and pheromone update are executed as separate sequential steps. This is no longer the case, when taking the local pheromone update of ACS into account. In ACS, the whole population constructs their tours at the same time. I.e., the whole population does exactly one step (extends its tour by one more city) before any ant makes the next step. This way, the choice an ant makes in a particular step of the tour construction can directly impact the decisions made by other ants within the same iteration.

Another innovation to be pointed out here is a speedup technique used in ACS, namely *candidate lists*. When choosing the next city during tour construction, the algorithm would in a first step consider only the *cand* nearest neighbors of the city. Only if all cities on that list were already visited, the remaining cities are considered. Due to performance gains related to the use of candidate lists, the concept was adopted in other ACO algorithms, among them the subsequently described MAX-MIN Ant System and Rank-based Ant System.

Pheromone update: The AS pheromone update has been significantly simplified (i.e., reduced in computational complexity) in ACS. Instead of evaporating pheromone on all edges in the network and depositing pheromone on all edges used by the ants, ACS uses only the best found tour. The superscript *bs*, which is commonly used in this context, is derived from the notion of the *best-so-far ant*. Accordingly, T^{bs} refers to the shortest tour discovered so far. Equation 2.13 illustrates the global pheromone trail update as typically implemented in ACS.

$$\tau_{ij} = (1-\rho) \cdot \tau_{ij}^{p} + \rho \cdot \Delta\tau_{ij}^{bs} \qquad \forall (i,j) \in T^{bs} \tag{2.13}$$

It should be noted that $\Delta\tau_{ij}^{bs}$ - again derived using formula 2.6 - is not added at its full value. The evaporation factor ρ is effectively used as a learning rate, which permits for a less abrupt transition between pheromone levels.

An interesting effect of the proposed changes compared to AS is the implicit introduction of lower and upper pheromone bounds for all connections between cities. While equation 2.12 ensures that no edge's pheromone level drops below τ_0, the ACS' global pheromone trail update (i.e., equation 2.13) implicitly sets $1/C^{bs}$ as an upper bound for the pheromone values. For more details on this reasoning, the reader is referred to the book by Dorigo and Stützle [22].

2.3 MAX-MIN Ant System

Stützle and Hoos [74] proposed another variant of AS in 1996. Its name MAX-MIN Ant System (MMAS) is derived from the fact that it enforces a permissible value range for the pheromone values (i.e., it explicitly defines a maximal and a minimal pheromone value). In the following, the algorithm's differences compared to AS will be examined step by step. Two new algorithm parameters and several new variables will be used in this context:

g_τ	factor determining the size of τ_{min} relative to τ_{max} for setups with local search
λ	sensitivity of the λ-branching factor

avg	average number of different choices available to an ant at each step while constructing a solution
$best$	alias for the ant, resp. tour used in the pheromone update
f_λ^i	λ-branching factor of city $i \in N$
f_λ^{avg}	average λ-branching factor of all cities $i \in N$
$f_\lambda^{restart}$	threshold with respect to f_λ^{avg} that in combination with other factors triggers a pheromone re-initialization in MMAS
ib	alias for the iteration-best ant, resp. tour
$iter_{min}^{stag}$	minimum number of iterations without improvement before a restart may be triggered
rb	alias for the restart-best ant, resp. tour
τ_{min}	minimal pheromone value permitted on any edge
τ_{max}	maximal pheromone value permitted on any edge
τ_{min}^i	lowest pheromone value on any edge incident to city $i \in N$
τ_{max}^i	largest pheromone value on any edge incident to city $i \in N$
S	set of all possible algorithm setups, resp. configurations
S_{ls}	set of $s \in S$ that use a local search method
$S_{\overline{ls}}$	set of $s \in S$ that do not use a local search method

Initialization: Two differences exist with respect to algorithm initialization. The first one is the use of a different initial pheromone level τ_0, as already seen for ACS. The second one, more of an innovation, is the fact that the pheromones are re-initialized during the run, whenever the search stagnates.

In contrast to ACS, which initializes the pheromone values with the smallest permissible value, MMAS uses its upper bound for the pheromone trails. Equation 2.14 describes how the value of τ_{max} is updated during an algorithm run.

$$\tau_{max} = \frac{1}{\rho \cdot C^{bs}} \tag{2.14}$$

As the best known solution decreases during a run, the algorithm slowly increases the highest permissible pheromone value. Equation 2.15 derives the algorithm's initial pheromone level by using a nearest neighbor estimate of the tour length.

$$\tau_0 = \frac{1}{\rho \cdot C^{nn}} \tag{2.15}$$

In the case of a re-initialization of the pheromone trails, all pheromone levels are reset to τ_{max}. The stagnation of the search, which is the precondition for the pheromone values to be re-initialized, is determined using the λ-branching factor [21]. If the overall average of this city-specific measure stays below a certain threshold for some time, a re-initialization takes place. A city's λ-branching factor reflects the number of incident edges that have a pheromone value exceeding the smallest value on any of the incident edges by some minimum quantity. The latter is determined based on the size of the smallest and the largest pheromone value incident to the city, as well as the value of the parameter $\lambda \in [0, 1]$. Equation 2.16 formally describes the determination of the λ-branching factor for a city i as the number of τ_{ij} fulfilling the respective condition.

$$f_\lambda^i = \left|\left\{\tau_{ij} | j \in N, \tau_{ij} \geq \tau_{min}^i + \lambda \cdot (\tau_{max}^i - \tau_{min}^i)\right\}\right| \qquad \forall i \in N \tag{2.16}$$

The variables τ_{min}^i and τ_{max}^i refer to the minimum, respectively maximum of the pheromone levels τ_{ij} on edges adjacent to a city i. When choosing a reasonably small value for λ, the cities' average branching factor (see equation 2.17) approaches the value 2.0, as the pheromone values converge.

$$f_\lambda^{avg} = \frac{1}{n} \cdot \sum_{i \in N} f_\lambda^i \tag{2.17}$$

This is due to the fact that in the case of search stagnation, most ants construct the same tour and accordingly, two incident edges of each city would receive by far the most pheromone. Thus, the pheromone values are re-initialized whenever this average value stays below a threshold $f_\lambda^{restart} > 2$ (see condition 2.18) without any improvement of the best known solution for a minimum number of iterations $iter_{min}^{stag}$.

$$f_\lambda^{avg} \leq f_\lambda^{restart} \tag{2.18}$$

Tour construction: With respect to this algorithm component, no differences to Ant System were introduced in standard MMAS. However, the same does not hold for a modified version of MMAS which will be characterized towards the end of this section.

Pheromone update: The MMAS pheromone trail update starts with the evaporation of pheromone from all edges. This is illustrated by equation 2.19, which at the same time handles the pheromone deposit:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij}^p + \Delta\tau_{ij}^{best} \qquad \forall i, j \in N \tag{2.19}$$

$$\Delta\tau_{ij}^{best} = 1/C^{best} \qquad \forall (i, j) \in T^{best} \tag{2.20}$$

$$\Delta\tau_{ij}^{best} = 0 \qquad \forall (i, j) \notin T^{best} \tag{2.21}$$

Formulas 2.20 and 2.21 determine the amount of pheromone that is added, which is effectively computed in the same way as previously done for AS (see equation 2.6). However in MMAS, different choices with respect to the tour receiving the additional pheromone (i.e., T^{best}) are possible. While, e.g., ACS always chooses the best-so-far ant's tour T^{bs}, MMAS also uses the iteration-best ant's tour T^{ib} as well as the restart-best ant's tour T^{rb} (i.e., the best tour found since the last pheromone reset) in some situations. The choice, which tour to update is typically implemented as a fixed schedule, choosing the former among others based the iteration counter *iter* and the fact whether or not local search is used in the present configuration.

While more complex schedules have been proposed in literature [22], the present work uses a rather simple approach. With local search, only the restart-best tour is used. Without local search, the latter is used only every 25 iterations, while the iteration-best tour is updated for the remainder of the time.

Another aspect to be considered with respect to the pheromone trail update is the maximal and minimal pheromone trail limits from which MMAS derives its name. The upper limit τ_{max} has already been introduced in formula 2.14 and it is enforced implicitly, i.e., the combined effect of evaporation and pheromone deposit ensures that it is never exceeded. In contrast to this, MMAS manually enforces the minimal pheromone level τ_{min} after each iteration.

Different formulas for τ_{min} are used based on whether local search is applied or not. In the local search case, equation 2.22 derives the lowest permitted pheromone level based on τ_{max}, a fixed factor g_τ, and the problem size n. For applications without local search, the relationship of the pheromone trail limits is more complex (see equations 2.23 and 2.24). More details on the underlying reasoning are provided by Stützle and Hoos [76].

$$\tau_{min} = \frac{\tau_{max}}{g_\tau \cdot n} \qquad \forall s \in S_{ls} \tag{2.22}$$

$$\tau_{min} = \tau_{max} \cdot \frac{1 - \sqrt[n]{0.05}}{\sqrt[n]{0.05} \cdot (avg - 1)} \qquad \forall s \in S_{\overline{ls}} \tag{2.23}$$

$$avg = \frac{cand + 1}{2} + 1 \tag{2.24}$$

A new algorithm variant: While the above describes standard MMAS, there is one more variant of MMAS that is examined in the present work. It makes use of the pseudorandom proportional action choice rule as applied in ACS. That means the ants use equations 2.10 and 2.11 instead of 2.3 and 2.4 to choose the next city during tour construction. Besides this difference, the two approaches are the same. This idea has previously been explored by Stützle [73], respectively Stützle and Hoos [75]. In the following, the adapted variant will be referred to as $MMASq_0$. The latter refers to the additional parameter q_0 that becomes necessary when using the pseudorandom proportional rule.

2.4 Rank-based Ant System

Rank-based Ant System (RANKBAS) has been proposed by Bullnheimer et al. [12] in 1999. As in MMAS, the implementation used in the present work sets τ_0 according to equation 2.15. However, the main idea underlying the development of RANKBAS is concerned with how the ants deposit the pheromone.

Pheromone update: To illustrate this idea in more detail, the previously described algorithms' update rules will be quickly revisited. In AS, all m ants deposit pheromone on their tours. Due to the nature of the pheromone update rule (i.e., each ant k deposits $1/C^k$ pheromone on the edges it used), small differences in tour length also lead to relatively small differences in the deposited pheromone. ACS and MMAS take a different approach. They use only one ant per iteration for the pheromone deposit (i.e., the best-so-far, the restart-best, or the iteration-best ant). While AS has a relatively small focus on good solutions, one may argue that ACS and MMAS overly fortify tours that are at least as good as the present iteration's best ant's solution while ignoring the others.

Bullnheimer et al. introduce an approach that tries to find a balance between taking solutions into account that are inferior to the iteration-best while still ensuring an elitist behavior of the algorithm. They rank each iteration's solutions based on their solution quality and have the best $w-1$ solutions deposit an amount of pheromone that is weighted depending on the individual rank. In addition, the best-so-far tour receives the highest amount of pheromone. Accordingly, a new user-provided parameter is required:

w	number of ants that deposit pheromone in RANKBAS

RANKBAS' pheromone update rule may look somewhat lengthy at first sight. However, it is easily derived as subsequently explained:

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij}^{p} + w\cdot\Delta\tau_{ij}^{bs} + \sum_{r=1}^{w-1}\left((w-r)\cdot\Delta\tau_{ij}^{r}\right) \qquad \forall i,j \in N \tag{2.25}$$

Essentially, the term for τ_{ij} is composed of three parts. The first is common to all previously characterized ACO algorithms. It reflects the pheromone evaporation. The second part of the sum reflects the pheromone that is added on the best-so-far tour. It is weighted by the parameter w, which is the highest weight used for any of the tours in the pheromone update. The sum in the formula's third part implements the pheromone deposit for the tours ranked on positions 1 through $w-1$. They receive weights in the opposite order of their rank, i.e., a tour ranked r is weighted with $w-r$. The values for $\Delta\tau_{ij}^{bs}$ and $\Delta\tau_{ij}^{r}$ are again computed according to equations 2.6 and 2.7. In this context, it should be noted that the index r used in $\Delta\tau_{ij}^{r}$ identifies an ant by means of its rank.

Many different algorithm parameters were introduced in this chapter. Subsection 4.2.1 gives an overview of the reference configurations for ACS, MMAS, MMASq_0, and RANKBAS. Changes to these reference configurations, which are performed as part of the computational studies presented in chapters 5 through 7, are described in the respective chapters.

3 Parameter Optimization

Practitioners as well as researchers working with heuristic methods are mostly aware of the importance of using ”good” parameter settings. An algorithm’s performance is typically crucially dependent on choosing parameter values that reflect general requirements of the used method, e.g., by factoring in interdependencies between different parameters. To achieve peak performance, some methods even require their parameters to account for particular characteristics of the considered problem instance. Thus, it may not be surprising to note that first research on how to set a method’s parameters is oftentimes published in close timewise proximity to the method itself. Interestingly, even rather complex approaches to parameter setting are sometimes developed only little later. With respect to the field of Evolutionary Algorithms (EAs), two noteworthy examples are Rechenberg’s 1973 work on adaptive mutation [60] or Rosenberg’s 1967 PhD thesis proposing the adaptation of parameters during the run of a Genetic Algorithm (GA) [62]. Since then, a large body of literature has evolved. In order to be able to clearly distinguish and categorize the different approaches, various classification schemes were proposed. Section 3.1 introduces a recent scheme by Eiben et al. [27]. Using its terminology, section 3.2 gives an overview of related work. After examining previous research in EAs in subsection 3.2.1, subsection 3.2.2 summarizes approaches that were proposed in the context of ACO.

3.1 Classification of Parameter Setting Techniques

Many different classification schemes have been proposed that allow to distinguish different classes of parameter setting techniques. While the development of new classification topologies is sometimes pursued by researchers as a research subject by itself, the present work utilizes the former more from a practical point of view, rather than evaluating a specific scheme’s advantages and disadvantages.

The present section briefly discusses the classification as proposed by Eiben et al. [27]. It has been chosen for two reasons. First, it is general enough to be applied to a broad context, i.e., its categorical structure is not limited to a specific application domain (e.g., by using strictly GA-specific dimensions). Second, it has been found to capture most relevant details in an ACO context, while utilizing a widely accepted terminology base.

In fact, the main reason for introducing a classification scheme in the present work is to provide a clear-cut vocabulary to be used - not only - in the subsequent literature review (see section 3.2). The use of a uniform vocabulary will prevent confusion about the meaning of specific terms in the given context and will allow for an unambiguous discussion of the different parameter setting techniques to be introduced hereafter. Eiben et al. start their classification scheme by splitting parameter setting techniques into two major groups. In a first step, they distinguish parameter tuning from parameter control.

3.1.1 Parameter Tuning

The term parameter tuning denotes all procedures that are executed offline, i.e., before the relevant problem-solving run of the algorithm takes place. Effectively, a method can be considered parameter tuning if it determines the parameter values a priori and then executes the algorithm with static parameter settings. This explicitly includes simple approaches like manual trial and error or full factorial test designs (i.e., running the algorithm with all possible combinations from a reasonably discretized set of parameter values to ensure capturing all relevant trade-offs).

There are also more sophisticated methods that fall into this category. A first group of approaches typically applies a more or less fully fledged problem solver (e.g., a genetic algorithm) to tune the primary (i.e., the problem solving) algorithm. In such scenarios, the genetic algorithm sets the parameters of the problem solving algorithm and executes a full run of the latter for each parameter setting it comes up with. The algorithm's performance with that specific parameter setting is then evaluated and used to guide the exploration of the parameter space towards better performing regions. Eventually, the GA returns the best-performing parameter setting it found, which can consecutively be used as a tuned setting for running the problem solving algorithm. Examples of such methods will be given in the next section.

Another class of rather sophisticated methods approaches the tuning problem from a more mathematical perspective, i.e., by means of a statistical analysis of the parameter space. One such method will be reviewed in subsection 3.2.3.

3.1.2 Parameter Control

In contrast to parameter tuning, parameter control includes all methods changing an algorithm's parameters in the course of its run. This is where the main focus of the present work will be. Eiben et al. further subdivide this second group of methods by asking a set of three questions:

- How is it done?
- Which evidence is used?
- What is changed by the technique?

These questions will be discussed further in the following.

How is it done? With respect to *how* parameter control may be put into practice, the authors propose three different categories: deterministic, adaptive, and self-adaptive approaches.

Deterministic parameter control comprises all methods that change parameters according to some predefined schedule, i.e., independent of run-specific (in some sense non-deterministic) developments of the algorithm execution. One can, e.g., think of the parameters as being changed as a function of the iteration counter. In an ACO case, the latter typically refers to the number of executions of the algorithm's main loop. This is preferable to CPU time, because the iteration counter does not depend on the particular hardware.

The second category is referred to as *adaptive* parameter control. In contrast to the previous group of methods, adaptive approaches take decisions (i.e., change parameters) based on characteristics of the algorithm run. One typical representative of this category are rule-based approaches. These change parameters whenever a run-dependent measure fulfills a specific condition, respectively rule (e.g., when it hits a fixed threshold or when it reaches a specific ratio compared to some other measure). A second type of adaptive procedure that is frequently encountered in literature are meta-algorithms that control the parameters of the problem-solving algorithm in the course of its run. As a similar case has been illustrated with respect to offline-tuning, it is important to note the difference. In the case of adaptive parameter control, the meta-algorithm performs an online-optimization of the parameters while the problem solving algorithm is executed. In the parameter tuning case, the additional algorithm was used to determine a good set of parameters offline, i.e., before the relevant run of the primary algorithm.

The third and last type of parameter control is considered *self-adaptive*. In this case, some of the problem's parameters are included in the algorithm's internal representation of potential solutions to the problem. Accordingly, while the algorithm evolves solutions to the problem at hand, it also evolves possible choices for its own parameters. As this concept may not be intuitive at first sight, it will be illustrated using a short example from the field of Genetic Algorithms. Readers not familiar with the basic concepts used in this context may have to refer to literature [33] [5] for further details. A Genetic Algorithm would typically encode a problem solution as a series of genes, e.g., $X_1X_2X_3X_4X_5X_6X_7X_8X_9X_{10}$. Adding further genes that encode algorithm parameters P^1 and P^2 could lead to something like $X_1X_2X_3X_4X_5X_6X_7X_8X_9X_{10}P^1_1P^1_2P^2_1P^2_2P^2_3$ where P^1 is encoded using two genes and P^2 using three, respectively. The additional genes would now be subject to the same evolutionary process (i.e., the application of reproduction and selection operators) as X_1 through X_{10}. Accordingly, while the algorithm evolves the solutions encoded in the individuals' genome, it also adapts the contained parameters. For ACO implementations, Self-Adaptive Parameter Control can be realized by using a second network encoding different parameter choices on its edges. Each time, before an ant travels the standard network representing the problem's solution space, it would travel the parameter network to choose its own parameters. The ant would consecutively use these parameters when constructing its tour on the TSP's network and when updating the pheromone trails. Pheromone deposit and evaporation on the parameter network could be realized similar to how it is done in standard ACO. For more details on how to implement Self-Adaptive Parameter Control, the reader is referred to the respective work which is referenced in section 3.2.

Which evidence is used? The second question raised by Eiben et al. aims at categorizing the type of evidence that is used by the mechanism. *Absolute* and *relative* evidence were chosen as the relevant categories.

Evidence is considered *absolute*, when the absolute size of a measure determines whether an action is taken (i.e., whether a parameter is changed) or not. Typically, this takes the form of a measure being compared to a fixed (absolute) threshold and a parameter being changed only if that threshold is reached. A practical example of absolute evidence is, e.g., the use of the progress rate (i.e., a measure for the distance from the optimum) as an indicator for algorithm convergence. It should be noted that the use of absolute evidence requires a strong intuition, respectively a reasonably good understanding of the algorithm's internal interdependencies. This is necessary as rules based on absolute thresholds essentially require the researcher to be able to predict reasonably well which ranges are desirable or undesirable for a specific measure, and how the undesirable state can be left by changing a particular parameter.

This necessity is somewhat offset when using *relative* evidence as trigger for parameter changes. In the latter case, one compares two or more measures reflecting different aspects of the algorithm's evolution relative to one another. This would, e.g., be the case when comparing the rate at which applying a specific operator improved the best known solution (i.e., successful applications relative to total applications) to the respective rate of another operator. By comparing several measures that are based on the algorithm's evolution, the researcher is at least partially relieved from the need to specify good or bad ranges for a specific measure. In many cases, it may be sufficient to see that the impact of a certain setting A was better or worse than the effect of an alternative setting B without having to specify hard thresholds in advance.

What is changed by the technique? This is the third question used by Eiben et al. to differentiate parameter control techniques. It aims at identifying the aspect of the algorithm that is ultimately subject to adaptation.

In the past, researchers have adapted various components, respectively parameters of Evolutionary Algorithms as well as Ant Algorithms. Starting with problem representation and objective function, different parameters with respect to mutation, crossover and selection have been adapted. Various approaches to adapt the population size used by the algorithm have been developed as well as many hybrid schemes (i.e., methods changing several algorithm components at once). For now, this list will not be detailed any further. However, section 3.2 will provide the reader with a series of examples for each of these categories.

The Scope of Change: Initially, Eiben et al. proposed a fourth classification criterion which was later abandoned for different reasons. In their earlier work they considered the *Scope of the Change* to further subdivide the previously illustrated categorical structure [42]. E.g., a parameter change could have an effect at the level of a full solution (i.e., impact a whole individual) or at the level of a solution component (e.g., a parameter affecting an operator's behavior only on specific genes). It could also impact the full population or have an effect that is reflected in the behavior of the environment (e.g., changes to the objective function). Finally, this fourth distinction has been dropped for two reasons. Firstly, in many specific cases it turned out impossible to clearly depict the scope of a change in an unambiguous way. Secondly, the scope was discarded as it was primarily a property of the parameter which is subject to adaptation, but not a property of the adaptation process itself. Thus, the change's scope will not be considered any further in the present work.

3.2 Literature Review

This section gives an overview of previous work done on parameter tuning and control. First, subsection 3.2.1 will give a general overview of methods that were developed in the context of Evolutionary Algorithms. It is followed by a subsection doing the same for Ant Colony Optimization. In general, the focus will be on parameter control rather than tuning, as this aligns closer with the experimental studies presented in subsequent chapters. Subsection 3.2.3 will briefly touch upon some general parameter setting concepts that are applicable to a larger range of algorithm types.

3.2.1 Prior Work on Evolutionary Algorithms

Previous work on Evolutionary Algorithms has been integrated into this literature overview for several reasons. Both EA and ACO are stochastic, population-based algorithms inspired by nature. This leads to the assumption, that some of the ideas developed for parameter setting on the one algorithm type may transfer to the other. At the same time, the field of EA had about twenty additional years to mature and thus promises to be a rich source of inspiration with respect to parameter tuning and control.

Generally, the term Evolutionary Algorithms refers to a family of algorithms. Its most prominent members, i.e., Genetic Algorithms and Evolution Strategies (ES), are reviewed in this subsection with respect to parameter setting approaches that were proposed. Readers that are not familiar with these algorithms are referred to the book on GAs by Goldberg [33] and to an article about ES by Beyer and Schwefel [9] as introductory literature.

Review papers that may be of general interest in this context are work by Hinterding et al. [42] on adaptation in Evolutionary Computation as well as a book chapter by Eiben et al. [27] on parameter control in EAs. De Jong [16] published a personal retrospect on the historical developments since the early days of EAs.

Somewhat similar to the referenced work by Eiben et al., the remainder of this subsection will be structured based on the parameter, respectively algorithm component that is subject to adaptation. Following the single components, approaches changing several parameters or components at once will be introduced. Each of these sub-topics will close with a brief subsumption on its relevance in the context of Ant Colony Optimization.

Adapting Problem Representation

The first aspect of Evolutionary Algorithms that is discussed here as a potential adaptation target is the problem representation. In EA terminology, the latter is typically referred to as the problem coding. The problem coding describes which aspects of the underlying mathematical problem are reflected, in which specific manner, in the character string representing an individual in the population. In other words, the coding determines how potential solutions are represented inside the algorithm.

However, when taking this thought one step further, it becomes clear that the question, which information to code and how to do it, is a lot more than just a simple design choice which the algorithm practitioner can take the one or the other way without major consequences. In practice, it turns out that with the choice of a specific coding, the developer implicitly limits the possible choices on other algorithm components. At the same time, maybe most importantly, it sets the ground for a predictably bad or surprisingly good performance of the algorithm.

Early work with respect to changing the problem representation was done by Shaefer [67] in 1987. He proposed the *ARGOT strategy*, which pertains to the context of classical GAs. The latter typically discretize a problem's real-valued variables by mapping them onto bit strings. Shaefer's method dynamically changes the value range, respectively the resolution of the variables' binary representation. While this may not be intuitive at first sight, range and resolution of variables may be valuable instruments to adjust the algorithm's search behavior. E.g., using a fine (i.e., detailed) resolution of variables when the algorithm is close to convergence can be helpful to allow it to discover the highest peak in a locally optimal area of the solution space. However, to discover such areas, it may be more profitable to quickly screen large parts of the solution space with a more strongly discretized representation of the relevant variables. Another strength of the ARGOT strategy is its ability to center a variable's binary representation on the values effectively represented in the population. This allows to reduce the number of bits required to encode a variable when its range and resolution are decreased.

Delta Coding by Whitley et al. [79] changes the aspect of the problem that is actually coded. Instead of explicitly coding possible solutions as it is typically done in EAs, the algorithm evolves a set of modifications (i.e., deltas) of a particular interim solution. I.e., the individuals encode only the deviations from this interim solution - not the solution itself. The interim solution is updated (i.e., replaced by the solution implicitly represented by the best individual) every time the population diversity drops below a certain threshold.

Another branch of research is centered on the building block hypothesis proposed by Goldberg [33]. Therein he argues that certain alleles, respectively genes within an individual's genome contain information that is related (i.e., they form a building block). Starting from there, he explains that in order to successfully propagate the knowledge contained in the related genes, the respective genes need to be exchanged together, i.e., applying operators such as crossover should ideally not break building blocks into peaces. To achieve this goal, Goldberg et al. propose the *Messy GA* [36], respectively the *Fast Messy GA* [35]. It dynamically repositions the bits in the coding in order to have all genes that are part of a particular building block close, respectively adjacent to one another. In return, this reduces the frequency with which valuable information gets lost, respectively building blocks get broken into pieces by crossover. With a similar goal, Paredis [56] developed a GA approach based on two populations. While he uses one population encoding the variables' ordering on the GA string, the second population carries the variables' values. Both approaches were shown successful in identifying building blocks, respectively in grouping interdependent genes closer together.

With respect to ACO, adapting the problem representation would come down to dynamically changing the meaning of edges and nodes in the network representing the TSP. While the problem representation generally used in Ant Algorithms is to some degree intuitive, it is not the only feasible approach. Nevertheless, most straightforward alternatives would typically be of a similar static nature and not involve any adaptive aspects. By the time this document is created, the author is not aware of any research examining related ideas in ACO. For future research, it may be an interesting idea to, e.g., investigate the application of Delta Coding to Ant Algorithms. As a first major step, this would require to find a smart way to map changes to a given tour onto a network structure. The present work however, does not consider adaptations of the problem representation any further.

Adapting the Objective Function

Another aspect of Evolutionary Algorithms that has been adapted by researchers is the algorithm's objective function. While this may not be a straightforward idea with respect to many frequently examined optimization problems, possible motives for adapting the objective function become rather clear when considering the context of constraint satisfaction problems. In this specific case, i.e., when searching for a solution satisfying a set of given constraints, it is a widely used approach to enforce constraints by means of penalty terms in the objective function. Whenever a constraint is violated, the respective penalty term worsens the objective function value that would otherwise be achieved by the considered solution.

As there are typically constraints that are harder to satisfy than others, most such methods use weights on the penalty terms to account for these differences. However, when exploring different subregions of the solution space in the course of an algorithm run, different constraints may be hard to satisfy at different stages of the run. Thus, the constraints' weights, i.e., a part of the objective function, are a potential adaptation target.

Eiben and Ruttkay [28] use a self-adaptive approach to assign the penalty weights. Every iteration, the weights are subject to the evolutionary process. Sawing by Eiben and Van der Hauw [25][29] periodically adjusts the weights. High weights are assigned to those constraints that are not satisfied by the population's best individual.

With respect to the ACO variants as considered in the present work (see chapter 2), constraint-based adaptations of the objective function did not appear reasonable to the author. This type of adaptation seemed promising primarily for optimization problems dealing with more complex constraints than the TSP, respectively with such that are harder to satisfy. Alternative adaptations as, e.g., punishing specific tours in order to get the algorithm out of local minima, were discarded in the given context as they seemed to be distorting with respect to the nature of ant algorithms. Same as for adapting the problem representation, there appeared to be no active research on adapting the objective function in ACO by the time this work has been completed.

Adapting Mutation

When talking about adapting mutation in Evolutionary Algorithms, it is important to distinguish which type of EA is considered. This is necessary as having been developed largely independent, Evolution Strategies and Genetic Algorithms use different mutation operators with different parameters that could be subject to adaptation.

Considering a classical implementation of an *Evolution Strategy*, the algorithm typically works on a real-valued alphabet. A Gaussian mutation is performed every iteration on all genes of the genome. What is typically changed about the ES mutation operator is its step size, respectively the Gaussian mutation's standard deviation. In more recent approaches, the step size typically varies for each gene in the genome, i.e., based on the nature of the information represented by a gene, it can be of individual size.

The first and maybe most prominent approach for adapting the mutation step size in ES was proposed by Rechenberg [60] in 1973. His *1/5th rule*, an adaptive method, is based on the fundamental assumption that to obtain nearly optimal performance for a (1+1)-ES (i.e., a basic ES), about one fifth of all mutations should be successful. I.e., to obtain an optimal balance between exploration and exploitation, about 20% of mutations should produce offspring having a higher fitness than the parent individual. If the success rate is lower, it is assumed that mutation introduced too much variation and in return, the step size is decreased. When experiencing a higher success rate, Rechenberg concluded that the algorithm is moving towards a local optimum with steps that were too small and thus, the step size should be increased. At its time, the 1/5th rule was widely applied in the ES community.

In 1981, Schwefel [66] proposed a self-adaptive method to let the evolutionary process modify the mutation step size. Based on the method's success and some limitations [9] of the 1/5th rule, self-adaptive ES (SA-ES) took over as the standard procedure for parametric optimization for quite some time [6].

In 2001, six years after publishing first underlying concepts, Hansen and Ostermeier [40] [39] introduced the current state of the art for adapting mutation in ES. They proposed a new rule-based (i.e., adaptive) method called *CMA-ES*. As opposed to earlier methods, it took the correlation between different genes in the genome into account. In the course of its run, it updates a respective covariance matrix of the multivariate Gaussian distribution used for the mutation. The method has been subsequently improved by Hansen et al. [38] in 2003.

The following paragraphs will take a look at the adaptation of *mutation in Genetic Algorithms*. Compared to Evolution Strategies, it has to be noted here that the role of the mutation operator in genetic algorithms is of a somewhat different nature. Having crossover at its side as a means to mix different individuals and to thereby create new combinations of genes from the population's gene pool, it is no longer suitable to mutate each and every gene of an individual at every iteration. Considering the case of a classical GA, which by default uses a binary alphabet, mutating all genes would even be useless. That is because flipping all bits from one to zero or vice versa would essentially invert the individual as a whole at every iteration. Accordingly, the mutation's aim to introduce a useful degree of diversity into the population cannot be met this way. Thus, a different approach to mutation has been chosen for Genetic Algorithms. In a GA context, the first question is whether at all a gene shall be mutated, rather than how large the change should be. Therefore, the relevant parameter of a GA's mutation operator is the probability with which a mutation is performed on a specific gene. The following paragraphs will examine methods that change this parameter during the algorithm run.

Similar to ES, the idea of adapting mutation in GAs goes back to the early days. In 1975, Holland [44] proposes to change the mutation-rate during a run as a function of time (i.e., deterministic). However, one and a half decades should pass until this idea underwent a more detailed examination. In 1990, Hesser and Männer [41] published results of successfully setting the mutation probability with a time-dependent function.

In 1992, Bäck [5] introduced a GA variant with self-adaptive mutation probability. Bäck's method was improved by Smith [68] in 2003 with respect to shortcomings that caused the original implementation to stagnate around mutation probabilities of zero. While Bäck's self-adaptive method used the algorithm's normal mutation probability to perform mutations on the genes encoding this very same rate, Smith used a fixed mutation rate on these specific genes. Thus, a mutation rate close to zero does no longer cause the algorithm to stall in a state without mutation.

When reviewing mutation adaptation in ES and GAs, two more aspects may be noteworthy besides the previously mentioned technical differences. Firstly, since the 1/5th rule was published in 1973, ES practitioners mostly used the respective state-of-the-art method to adapt the mutation step size of their algorithms. Compared to that, the GA community is only modestly applying any type of adaptation to their mutation operator. If at all, it is mostly self-adaptive methods that are used to change a GA's mutation rate. Secondly, the evolution from Rechenberg's 1/5th rule over Schwefel's self-adaptive SA-ES towards Hansen and Ostermeier's CMA-ES illustrates another interesting aspect. While the self-adaptive SA-ES was able to outperform Rechenberg's simple adaptive method, CMA-ES in return was able to improve the results from SA-ES. That implies that simple intuitive thoughts like "with self-adaptation, the algorithm will choose what's best for it" cannot be concluded to "self-adaptation is generally superior to rule-based adaptive approaches". If the researcher is able to identify the right information to change parameters in a rule-based manner, these methods can very well be competitive.

With respect to ACO, there is no real equivalent of the mutation operator in Evolutionary Algorithms. It is rather a multitude of parameters such as ρ or the combination of α and β that make the algorithm explore new regions of the solution space. Accordingly, adapting the mutation rate will not be considered any further in the remainder of this work.

Adapting Crossover

Crossover is the main recombinative operator used in Genetic Algorithms, i.e., it mixes the genome of different individuals from the present population to form new individuals. Therefore, it disjoints the former at several positions and consecutively merges the parts into new genomes.

A first approach to adapt crossover in ES was developed by Schaffer and Morishima [63] in 1987. They introduced a self-adaptive punctuated crossover operator. During the run, this operator changed the number of crossover points as well as their location.

Four years later, Davis [15] proposes an adaptive method that chooses a specific crossover operator based on its past performance. To determine the latter, he measures the improvement of individuals that were created using a specific operator over the previous population best. This reward is then propagated in a diminishing manner to the operators that were used to create the specific individual's ancestors. At fixed intervals, i.e., after a fixed number of generations, the algorithm redistributes 15% of the operator probabilities relative to the operators' performance measures.

In 1995, Smith and Fogarty [69] [70] proposed another self-adaptive method somewhat similar in spirit to the above method proposed by Schaffer and Morishima. Again, the number and position of crossover points is subject to adaptation. The difference is however that Smith and Fogarty no longer restrict their crossover operator to mixing two individuals. In a first step, their *LEGO* algorithm determines in a self-adaptive manner, which genes are linked, respectively are part of the same "block". The new crossover operator works by filling a genome from the left to the right. All individuals that have a gene block starting at the considered position compete for contributing the respective block based on their fitness. Accordingly, in an extreme case, the resulting individual may be composed of as many parents as there are genes in the individual (if the population size permits).

Similar to the case of mutation, there is no operator in ACO that explicitly enforces the mixing of solutions in ACO. Accordingly, the above illustration primarily served the idea of completeness and will not be extended in what follows.

Adapting Selection

Depending on the type of algorithm that is considered, selection operators may be encountered at two different stages of an algorithm iteration. The first selection operator is typically used to select individuals from the present population that will be used to generate offspring (i.e., *parent selection*). In a second selection step, the algorithm chooses the individuals among parents and / or offspring that will make it to the next iteration (i.e., *survivor selection*). Across the history of Evolutionary Computation, many different ideas on how to select the respective individuals in an appropriate manner were developed. Some algorithm variants even eliminate the need for one of the types of selection, e.g., by exclusively transferring the offspring (i.e., none of the parents) to the next generation. Other algorithm variants did even introduce further selection steps, e.g., as part of other operators such as local search.

Unrelated to the number and type of selection operators in an algorithm, there is a major intuition that unites most of the research community: A low selection pressure (respectively an only slightly elitist selection operator) in early stages is helpful to have the algorithm explore larger parts of the search space. As the algorithm converges, the selection pressure should increase (i.e., selection should become more elitist) to focus the search on the promising regions of the same.

Along these lines, many researchers focused on Boltzmann selection [54], i.e., a selection scheme changing the selection pressure based on a given criteria. The general principle is similar to Simulated Annealing (SA) [1]. Research on adapting the selection pressure is typically concerned with defining an appropriate *cooling schedule*, i.e., with designing the measure based on which the selection pressure should change in the course of the run. In terms of implementation, Boltzmann selection typically works by applying a scaling function to all objective function values before these are communicated to a respective selection method. By increasing or decreasing the relative difference between the individuals' objective function values, the method is able to steer the advantage of fitter individuals over others (i.e., effectively, the degree of elitism).

Early work on Boltzmann selection was done by De la Maza and Tidor in 1991 [17] and 1993 [18]. Further Boltzmann selection schedules (i.e., effectively control methods adapting the used selection operator) were proposed by Mahnig and Mühlenbein [55], Dukkipati et al. [23], and Yunpeng et al. [81]. It should be noted that only the method by Mahnig and Mühlenbein can be considered Adaptive Parameter Control, as both other approaches are deterministic.

In 2006, Eiben et al. [24] propose a self-adaptive scheme governing their GA's selection pressure. What makes their approach different from most other self-adaptive methods is that they use a local approach to adapt a global parameter. While the genes encoding the self-adaptive information can essentially take different values for each individual, the selection pressure is one common value for the whole population. Eiben et al. solve this contradiction by implementing a voting mechanism in which each individual contributes a small part to the global selection pressure. The latter is implemented as the tournament size of a classical tournament selection. Even though not necessarily intuitive, the authors show the applicability of their method. In a later paper, Vajda et al. [78] compare the approach to choosing fixed selection operators, respectively to other approaches adapting the same. While their method was presented as slightly superior considering the average performance over all considered classes of test instances, it was superior to all other methods only in one out of six classes. Problem-specific fixed settings seem to be superior in many cases.

Krasnogar and Smith [48] propose a somewhat different application of Boltzmann selection in the context of memetic algorithms (i.e., hybrids of EAs and local search). Their algorithm uses a Boltzmann scheme to decide whether or not to accept solutions produced by the used local search. Whenever the diversity of the population's fitness values is considered too high, the algorithm uses a more elitist selection in the local search (i.e., a higher selection pressure). On low fitness diversity, the selection pressure is decreased.

Even though there has been a fair amount of research on adapting the selection operator in EAs, it needs to be noted that in everyday algorithm usage, none of the adaptations is applied on a regular basis.

Similar to adapting mutation, there is no exact equivalent to the EA selection operator in ACO. While q_0 in combination with α and β governs the tour construction (i.e., the *selection* of single steps in the tour), this does not really resemble the previously sketched concept.

However, the selection step choosing the ants which are permitted to deposit pheromone at the end of an iteration shows some similarities. This particularly pertains to RANKBAS, which explicitly allows more than one ant to deposit pheromone on its tour. By adjusting the maximal rank that permits an ant to deposit pheromone, one could effectively control the balance of quality vs. diversity in the pheromone update. In some sense, one would control the degree of elitism used in the search. In a similar manner, one could create more complex update schedules for MMAS or enrich other AS variants with such concepts. Even though the above idea appears applicable, the author is not aware of any prior work using similar concepts in an ACO context. As it could not be examined further in the scope of this thesis, the idea is left as a potentially promising topic for future research.

Adapting Population Size

With the adaptation of the population size, a parameter that has a clear equivalent in ACO is the next to be examined in this review.

Smith, respectively Smith and Smuda in 1993 [71] and 1995 [72] were the first to change a GA's population size while the algorithm is executed. Their intent was to replace the population size parameter of the GA by something that had more meaning even to inexperienced GA users. Using population sizing theory by Goldberg [33], they proposed a method that determined the optimal population size from a user-specified *expected selection loss* in combination with some information about the state of the search. However, it turned out that the solution accuracy suggested by setting the selection loss to a specific value did not materialize. While there are several reasons for this, the work's real importance lies in its pioneering character with respect to population size adaptation, rather than in its actual results [53].

In 1999, Lobo and Harik [51] proposed what they titled a *parameter-less GA*. Their primary intent was to relieve the user from setting the algorithm's parameters without pursuing peak performance as a main objective of their work. With respect to selection rate and crossover probability, the relieve is enforced by setting them to fixed values for which GA theory as well as previous work predicted good performance. As GAs seem relatively robust with respect to changes in these two parameters [37], this was considered reasonable. What is more interesting with respect to adaptation is how they handled the population sizing parameter. Their approach is fundamentally based on the assumption that a GA's solution quality grows with population size. I.e., a specific size is either large enough to reach a desired level of solution quality or a larger population is needed. To be able to access results for different population sizes, the algorithm evolves several populations in parallel. The idea is to evolve a small population several generations before the next bigger population is allowed to evolve a single generation. Whenever a large population is able to catch up in solution quality with a smaller one, despite its lower number of iterations, the smaller population is discarded as being too small and the algorithm run continues without it. This way, the method effectively controls the used population size by explicitly narrowing down the set of evolved populations. However, some restrictions need to be noted about this adaptive approach. Not only is the size of the smallest evolved population strictly increasing (i.e., it never decreases); the largest considered population size also gets fixed when choosing the initial set of populations.

Six years after Harik and Lobo, Yu et al. [80] propose another method based on Goldberg's population sizing theory. Determining gene-linkage (i.e., the building blocks) on the fly, they are able to estimate situation-specific values for all relevant parameters of the respective population sizing formula. Sticking more closely to the reasoning originally underlying Goldberg's formulas they were able to show adaptive behavior of the population size. According to the authors, the method has been shown to be robust and efficient in finding what theory considers the optimal population size.

One method frequently discussed in literature has been developed by Arabas et al. [3] in 1994. Instead of varying the population size explicitly, they effectively replace this parameter by a new concept. They introduce the notion of an individual's lifetime to the GA. Whenever a new individual is created, it is assigned a number of generations for which it will stay in the population. Individuals with a high fitness receive higher lifetime values than non-fit individual. Arabas et al. argue that this is a good idea as it allows fit individuals to mate more often than others, which consequently leads to good genes being passed on more frequently than bad ones. While the authors report good results, the algorithm appears to be somewhat sensitive to a newly introduced parameter, i.e., the fraction of the present population size determining how many individuals are allowed to reproduce every iteration [26]. Depending on this parameter's value, the algorithm seemed to frequently stagnate in performance at extremely high population sizes. It should be noted that the use of the lifetime concept not only eliminates the population size parameter, it also relieves of the use of survivor selection and, in the form applied by Arabas et al., replaces parent selection by a simple random selection mechanism.

Fernandes et al. [31] [30] propose a modification to the above lifetime idea. They extend it by different forms of mating restrictions. I.e., they prevent the mating of related individuals (i.e., they prevent incest) and they examine preventing the mating of similar individuals. Another related method is introduced by Bäck et al. [7]. To prevent the previously mentioned uncontrolled growth of the population size, they apply the lifetime idea in a steady state context, i.e., they fix the number of individuals that are allowed to reproduce every iteration instead of using a relative measure based on the present population size. In addition, the best individual is immortal until a better one is discovered. Lobo and Lima [52] show 6 years later that after an initial phase in which the population size is adapted, the used steady state approach stagnates around a maximal population size that is based on the maximal lifetime assigned to individuals.

Just recently, Laredo et al. [49] proposed another approach to adapt a GA's population size. In contrast to all previous methods, they use a deterministic schedule to decrease the population size during the algorithm run. First results indicate improved convergence behavior (i.e., less iterations are needed to converge to the optimum).

It should be noted that besides the previously listed population size adaptation methods, further ones will be listed in the consecutive paragraphs. These were separated from the above as the respective algorithms also change other parameters at the same time. For a more detailed review of population sizing methods in a GA context, the interested reader is referred to work by Lobo and Lima [53].

Considering the mentioned work based on Goldberg's population sizing theory, this can barely be translated to ACO for which only little such theoretical groundwork exists to date. While, e.g., Pellegrini et al. [57] undertake an approach to explain MAX-MIN Ant System's parameter from a theoretical perspective, a lot more work needs to be done towards a profound understanding of Ant Algorithms and their parameters.

The general intuition underlying Harik and Lobo's parameterless GA (i.e., the larger the population, the better the results that can potentially be reached) is not very GA-specific and actually translates to some variants of ACO. However, as the proposed method focuses on facilitating algorithm usage at the cost of computational efficiency, the idea does not align with the purpose of this work and is thus not investigated further in its scope.

The lifetime idea by Arabas et al. is not applicable in an ACO context. While individuals in EAs carry a specific solution within them, this is not the case for ants in ACO. Accordingly, it does not make sense to use ant lifetimes to vary the population size in this context. An interesting idea that is nevertheless not within the scope of the present work could be to assign lifetimes to the best known solutions used for the pheromone update, i.e., to effectively introduce more powerful pheromone updating schedules.

Last but not least, deterministic schedules as used by Laredo et al. are very well applicable to ACO and are subject to further investigation in chapter 6.

Adapting Several Parameters Simultaneously

In the following paragraphs, several methods adapting more than one parameter simultaneously will be briefly reviewed.

Schlierkamp-Voosen and Mühlenbein [64] [65] adapt the population size of a set of subpopulations to reward the performance of their individual search strategies. I.e., based on how well the respective recombination and mutation operators used in a subpopulation perform, the subpopulation gets more, respectively less individuals. This implicitly reflects an adaptive approach changing the allocation of CPU time to different operators in the course of a run. Their first approach used a fixed total number of ants and changed the different population sizes by effectively reassigning the available ants to other strategies. The method proposed two years later no longer had this limitation and was able to vary the subpopulations' sizes independent from a global limit. It also allowed for a strategy-specific *consumption factor* to account for the different population sizing requirements which different operator combinations respectively different subpopulations may have. While the latter may be a reasonable thing to be done, it should also be noted that it effectively introduces one new (potentially sensitive) parameter per search strategy. In both versions of their algorithm, Schlierkamp-Voosen and Mühlenbein perform an exchange of the best solution between the subpopulations.

In 1996, Hinterding et al. [43] propose a method they call *SAGA*, which is an abbreviation for self-adaptive Genetic Algorithm. While it is self-adaptive with respect to mutation, at the same time it applies adaptive concepts to optimize the population size. The authors use three populations (i.e., $P1$, $P2$, $P3$), which are evolved in parallel. With respect to their size $P1$ is generally the smallest, while $P3$ is the largest. Two sets of rules are applied at regular intervals (i.e., after a fixed number of iterations) to adapt the size of the three populations as the algorithm is executed. The first set of rules aims at maintaining a sufficient diversity between the population's fitness values, whereas the second one aims at keeping the middle-sized population $P2$ the best performing one. The first set increases the difference in size if two populations come too close to each other with respect to their best individual's fitness value. The second one makes sure, the best population size value is surrounded by smaller and larger test values so that the algorithm is able to notice when adapting the number of individuals in either direction may be profitable.

In the same year, Lis and Lis [50] propose an algorithm that also evolves a series of populations in parallel to change mutation and crossover rate as well as population size in an adaptive manner. For each of the mentioned parameters, a small number of different values are defined (e.g., *low*, *med*, and *high*), which get assigned to the used set of populations. After a fixed number of iterations, the algorithm determines the parameter level that on average performed best, i.e., the parameter value that produced the highest average fitness considering only the best individual from each population using that value. Based on how this value compares to the previous best allocation of that variable, the respective variable's value may be increased or decreased for all populations.

Under the heading "Adapting Population Size", Bäck et al.'s work from 2000 [7] has already been mentioned as an application of the lifetime idea by Arabas et al. Here it is considered once more, as Bäck et al. examine several different configurations of their method. Besides adapting only population size, they also investigate self-adaptive mutation and crossover operators as well as combinations of the two. When comparing these different variants, the authors were able to show one especially interesting result. Adapting only the population size by means of the lifetime concept turned out close in performance to the combination of the named approaches. Using only self-adaptation on mutation and crossover rates however, performed significantly worse.

Assessing the relevance of the above for Ant Colony Optimization, the approach by Schlierkamp-Voosen and Mühlenbein (i.e., assigning population size based on search strategy reward) in this generalized form is for sure applicable to operator selection in Ant Algorithms. As illustrated in the next subsection, first applications have already been developed. Similarly applicable is the concept of Hinterding et al. to evolve three populations of different size to be able to adapt $P2$ to the optimal population size. Holding some interesting aspects from a conceptual point of view, the method is not considered any further in the present work due to its excessive use of computational resources. An idea similar to the one proposed by Lis and Lis and applied to an ACO context will be reviewed in the next subsection. The lifetime based approaches have already been commented on when evaluation population size adaptation.

3.2.2 Prior Work on Ant Colony Optimization

This subsection will introduce the reader to previous work on parameter tuning, respectively parameter control that has been published in the context of ACO. Even though the field is relatively young compared to Evolution Strategies or Genetic Algorithms, already a considerable amount of literature exists. While research on adaptive EAs dates back several decades into the 20^{th} century, most methods dealing with parameter setting in ACO, besides manual and some automated tuning approaches, have been developed in the new millennium.

Parameter tuning in ACO

As for many other algorithm classes, some research has been published on determining static parameter settings that perform well. As the present work's focus rather lies on parameter control, the following paragraphs will present only one specific parameter tuning approach from which interesting conclusions for the former may be derived.

The first method to be illustrated here deals with offline tuning of an ACS algorithm and has been proposed by Botee and Bonabeau [11] in 1998. In essence, all parameters relevant with respect to ACS, and even additional ones compared to those introduced in section 2.2, are optimized by means of a GA. While Botee and Bonabeau are able to present reasonably well performing parameter values for the used TSP instances, the more interesting aspect about their work may be some preliminary results which they report in their conclusion. They briefly explain the concept of a new algorithm variant using three subpopulations with individual values for q_0 as well as individual population sizes. This time, the GA evolves only the three values for q_0 and two values defining the split of the available ants on the three subpopulations. The authors' preliminary results indicate a significantly improved convergence. However, it needs to be noted that the used test instances were relatively small and that no local search has been applied.

Revisiting the idea underlying the second method proposed by Botee and Bonabeau from a more general perspective, they effectively simulate the division of labor as it can be observed in real ant colonies [61]. This notion of having different *casts* within the colony may be worth further exploration. Nature for sure has its straightforward reasons for combining different types of ants (e.g., guards, workers, a queen) in a colony. Even if some of these may not apply to the artificial ants in ACO, introducing a certain degree of diversity into the population's parameters may very well have a positive impact on the exploration of the search space, respectively the observed solution diversity.

Adaptive ACO using Meta-GAs

The next two methods apply GAs in an online context, i.e., for adaptive parameter control. Pilat and White [58], as well as Gaertner and Clark [32] both propose a hybrid of ACS with a Genetic Algorithm, the latter setting the parameters of the former while it is running.

Pilat and White use the GA to control β, ρ, and q_0 for 20 individuals. Every iteration, the algorithm selects four parameter settings which get assigned to (again four) ants which use them to build their tours. The two best-performing ants generate offspring (i.e., new parameter settings) in the GA. The obtained results indicate a possible decrease in the solution quality's variability as well as a higher convergence speed in early stages of the run when using larger instances (e.g., 783 cities). However in most cases, standard ACS seemed superior with respect to the final solution quality.

Gaertner and Clark chose to adapt only β and q_0. Their results indicate a performance comparable to standard ACS while relieving the user from setting, respectively tuning the two parameters manually. One particular aspect that should be considered when comparing the work by Pilat and White to the one by Gaertner and Clark is the way they combine GA and ACS. The former authors pretty much use a standard GA and combine it with an ant algorithm that quite strongly deviates from the initial version proposed by Dorigo and Gambardella [21]. Gaertner and Clark take the opposite approach. While more or less using a standard ACS, they make quite an effort to configure the GA, finally taking a series of non-standard design decisions on the GA side.

Adaptive ACO Using Subpopulations

The next two approaches to be reviewed have both been proposed in 2008. Besides this detail, however, a more relevant aspect they share is that they both use subpopulations in an adaptive parameter control context.

Anghinolfi et al. [2] adapt β and q_0 of an ACS implementation (for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times) by means of a neighborhood search. The concept requires the use of $1 + 2 \cdot c$ subpopulations, c being the number of adapted parameters. While one population uses the best performing parameter values from the last iteration, there are two additional populations per adapted parameter. One increases and one decreases the value in question while keeping all others fixed. After each iteration, the parameter setting that performs best becomes the new center of the adaptation. The adaptation ends once and for all when there is no change in the best known parameter values for ten consecutive iterations. The authors compare the new adaptive approach to the non-adaptive ACS variant underlying their new method and contrast the solution quality to previous results. They report a series of improvements over previously best known solutions with both algorithms. While the adaptive approach is presented as superior to the non-adaptive one, the documented experimental setup suggests that this comparison may have been a rather biased one. On the one hand, the authors used non-standard parameter settings together with several modifications to standard ACS behavior. These may, at least in theory, make an adaptation rather necessary than profitable by itself. On the other hand, a somewhat randomly chosen termination criterion leaves room for speculation whether the non-adaptive method may have been terminated before reaching convergence. Ending all runs after 100 iterations without improvement effectively led to the adaptive method running about two to four times as long as the non-adaptive one. An interesting additional result discovered by Anghinolfi et al. is the following: The adaptive method decreasing β in the course of a run effectively confirms the widely accepted intuition that the relevance of heuristic information decreases as the algorithm converges.

Kovarik and Skrbek [47] use a simpler version of an adaptive Ant Algorithm, this time based on a MAX-MIN Ant System. It builds on the idea of different castes within the population as previously used by Botee and Bonabeau. In their algorithm, they use ten castes, which each use a different β in $\{-1, 1, 2, 3, ..., 9\}$. The relative size of the castes is adapted based on their performance: If a specific beta value produces more improvements of the best known solution, the respective caste is enlarged relative to the others. How exactly this adaptation works is not explained in the paper. As their study is focused on the parameter β, Kovarik and Skrbek were able to illustrate clearly what has also been pointed out by Anghinolfi et al.: High values for β are needed in early stages of the run, while lower ones are more effective later on. The authors report results according to which their caste-based approach clearly improves over standard MAX-MIN Ant System. However, it needs to be noted that their standard configuration used $\alpha = \beta = 1.0$ which by itself makes an adaptation desirable as β has been chosen rather low. For future work, Kovarik and Skrebek propose various extension scenarios for the caste concept that include more complex caste definitions (e.g., including the use of local search), castes with separate pheromone trails, or even caste definitions that are evolving themselves.

Self-adaptive ACO

The first self-adaptive method is introduced to the field by Randall [59] in 2004. His ACS variant uses two separate networks with underlying pheromone trails. As in standard ACO, one of them corresponds to the considered optimization problem. The additional network is used to have the ants choose their values for β, ρ, ξ, and q_0. Before an ant builds its tour on the former network, it travels the latter to derive the parameter values it uses for the remainder of the iteration. Every time it chooses an edge in the additional network, it effectively assigns a specific value to one of its parameters. Except the global evaporation rate, all adapted parameters are ant-specific, i.e., each ant can use an individual value in its tour construction. For ρ, Randall simply uses the value chosen by the best-performing ant. Experiments were run for TSP as well as QAP (i.e., Quadratic Assignment Problem) instances. For the TSP, the algorithm did not show consistent improvements over standard parameters. The adapted parameters seemed to converge after about 100 iterations. Interestingly, on the QAP instances, things looked different. The self-adaptive ACS seemed to improve over the standard parameters in most cases and the self-adaptive behavior appeared to be more likely to continue throughout the run.

The most recent work on ACO to be included in this review was published by Khichane et al. [46] in 2009. They propose two self-adaptive approaches for the constraint satisfaction problem. The first one effectively applies self-adaptation to the two parameters α and β. To some degree, it resembles the previously illustrated method by Randall. The second method chooses a more complex adaptation scheme, which may be quite unintuitive on certain problem types, among them the widely used TSP. Nevertheless, the underlying idea is interesting and may prove valuable in other contexts: Instead of having each ant evolve exactly one value for each parameter, each ant evolves as many configurations as it needs to go steps on the network representing the problem. That essentially means that each ant uses an individual parameter setting in its first, its second, its third step on the network and so forth. For the TSP this does not yield any advantages as the ants start out at a random node of the network and thus, there is no well-defined, respectively generalizable meaning of what a specific decision in its x^{th} step may imply. This is different for other problems, where the first step always determines a specific variable's value, the second step corresponds to another specific variable, and so on. This way, the individual parameter values can, e.g., be matched to specific decision variables of the objective function. Whenever a value for that specific variable is chosen, ant- and variable-specific settings for α and β will be used. The reported results show the second self-adaptive method performing best on average, followed by the first self-adaptive method. A non-adaptive variant of the same algorithm ranked third. All algorithms were run for the same number of cycles. However, as the paper does not provide any total CPU times for this configuration, the given verdict on the self-adaptation schemes' performance behavior (i.e., the time vs. solution quality trade-off) may not be final yet.

3.2.3 Selected General Parameter Setting Concepts

After reviewing a series of specific applications of parameter tuning and control in the previous subsections, the following paragraphs will briefly introduce the reader to some more general approaches that may be worth investigating in the future. In contrast to most of the previously named methods they are independent from a specific algorithm type.

The first method to be reviewed here is called iterative F-Race and was proposed by Balaprakash et al. [8]. It is an iterative tuning approach based on F-Race, i.e., a method introduced earlier by Birattari et al. [10].

Every iteration, it uses a mathematical model to sample a series of parameter settings across the parameter space. By performing statistical analysis on the results obtained from running the primary algorithm (e.g., an ACO algorithm) with these parameter settings, it is able to consecutively adjust the used mathematical model. Effectively, this permits to concentrate the available computational power on the more promising regions of the parameter space.

As all tuning approaches, iterated F-Race produces a single parameter setting, which it considers best-performing and which can consecutively be used as a priori setting for the problem solving algorithm. One of the methods' main strengths is its easy extensibility for tuning arbitrary parameters. A drawback that is common to most parameter tuning approaches is that they consider the solution quality only at one fixed point in time, i.e., after terminating the problem solving algorithm. The algorithm's performance development before this point does not have any impact on the parameter values recommended by the method.

Three more methods will be briefly reviewed in this section. All three are adaptive and are typically used for operator selection. During an algorithm run, they evaluate the performance of different operators and use this information to determine how frequent each of the operators should be applied in the future.

The term operator as used here may be interpreted in a flexible manner. That means that it does not only include typical algorithm operators, such as different selection methods of a GA. It can also refer to specific parameter settings. As an example, one may consider an ACO algorithm on which one wants to switch between different values for β based on their performance, e.g., $\beta_1 = 1.0$, $\beta_2 = 5.0$, and $\beta_3 = 10.0$. All of the three subsequently summarized methods would derive a measure for the different β's performance, e.g., by counting the number of times a specific β improved the best known solution relative to its total number of applications. Another option could be to integrate the magnitude of the improvement in the performance measure. After each iteration, the respective method would use this information to determine the β to be used next.

The first operator selection method to be sketched here is *probability matching* [34]. Its main objective is to assign probabilities of applying a specific operator that are proportional to the operator's performance. To ensure that no operators get abandoned by the algorithm due to its poor performance in early stages of the run, a minimal probability applies to all operators. The major drawback of the above method is that it does not maximize the expected overall reward, i.e., the number of successful operator applications. Instead, it distributes the number of applications proportional to the past operator reward. If, e.g., β_1, β_2, and β_3 were successful in 10%, 60%, respectively 55% of their applications, probability matching would make only a small difference in the relative frequency of applying β_2 and β_3. However, in order to maximize the expected reward, it would be the correct choice to apply β_2 as often as possible.

This drawback is offset by a method called *adaptive pursuit* [77]. The latter method assigns a maximal probability to the best performing operator while the remaining operators are assigned a minimal probability. To prevent extreme algorithm behavior, a learning rate ensures a more smooth transition between operator probabilities.

A third group of operator selection methods are *multi-armed bandit* (MAB) approaches [4]. Similar to adaptive pursuit, they try to balance the two objectives to exploit the best performing operator as much as possible while giving worse performing operators the chance to catch up. In addition to the operator reward as considered by the two previous methods, MAB approaches also take an operator's number of applications relative to the other operators' applications into account. Da Costa et al. [14] showed in 2008 that certain MAB variants perform better than the above two methods in dynamic environments (i.e., if the operator reward distribution changes during the algorithm run).

With respect to ACO, all four concepts touched upon in this subsection are applicable. Iterated F-Race, similar to other tuning approaches, may be used to derive good static parameter settings that could, e.g., be used as benchmark results for adaptive methods. However, in a first step, the present work aims at improving performance over standard settings proposed in the literature, i.e., by changing single parameters with respect to a fixed reference configuration. In future extensions of this work, comparisons of generalized adaptive methods to instance-specific, tuned parameter settings may be a reasonable step to consider.

The three mentioned operator selection methods could, in general, all be used to adapt ACO algorithm parameters during a run. While the above example was limited to the single parameter β, the same type of adaptation would also be possible for more complex search strategies determining several parameter values at once. E.g., search strategy 1 composed of β_1, ρ_1, and $q_{0;1}$ could compete against search strategy 2 composed of β_2, ρ_2, and $q_{0;2}$ and so forth. Based on how well the particular combination of parameter values performs, it would receive more, respectively less trials. Again, this may be an interesting approach to investigate in the future.

4 Experimental Setup

This chapter aims at allowing the reader to better understand and possibly reproduce the experimental setup used in this work. Three sections illustrate the properties of the examined TSP instances, the basic algorithm configurations that were used, as well as some general aspects of the setup.

4.1 Problem Instances

The experimental results presented in subsequent chapters were generated using TSP instances of seven different sizes between 100 and $6,000$ cities. For each size, three problem instances were considered. When referring to a specific one among the three instances of equal size, the following chapters will use identifiers consisting of the problem size n, a dash, and the index 1, 2, or 3. Thus, e.g., for 100 cities, the instances are referred to as '100-1', '100-2', and '100-3'.

All instances were randomly generated by using a uniform distribution to place the n cities on a square area of side length $1,000,000$. Table 4.1 gives an overview of all used instances and their best known solutions. For problem sizes 100 to $3,000$, the shown values are optimal. For the instances with $6,000$ cities, a state-of-the art heuristic derived numbers that are believed to be reasonably close to the respective optima. The shown solutions do not have any post decimal positions because the algorithm determines the edges' length as the Euclidean distance between two cities, rounded to the nearest whole number.

instance name	size (n) in cities	best known solution instance 'n-1'	best known solution instance 'n-2'	best known solution instance 'n-3'
100-*	100	7,492,995	7,497,272	7,761,716
200-*	200	10,693,978	10,110,134	10,288,632
400-*	400	14,767,181	14,203,962	14,468,493
800-*	800	20,831,321	20,627,409	20,665,436
1500-*	1,500	27,977,271	28,322,211	28,262,824
3000-*	3,000	39,862,065	40,197,266	39,577,798
6000-*	6,000	55,945,646	55,951,185	55,924,220

Table 4.1: Problem sizes and best known solutions of the examined instances

The present work typically uses the instances with index 1 (e.g., 3000-1) to perform a first analysis of a proposed methodology's effect on algorithm behavior. The remaining two instances are consecutively used to verify, whether these observations also hold for other instances (i.e., to check for instance-specific biases).

The restriction to only three instances per problem size has been made to limit the computational resources required to examine the effect of a particular method on all instances. This appeared reasonable because across the set of such random uniform Euclidean instances, the behavior of ACO algorithms is considered to be relatively stable with regard to instance-specific biases. Experimental results on the set of three instances per problem size confirmed this assumption.

Having said the above, executing further experiments to formally verify the subsequently shown results for other setups does not become obsolete by any means. However, the author believes that the used methodology is suitable to allow for first conclusions on the effect of ACO parameter optimization in a TSP context. More information on future options to extend this research is given in section 8.2.

4.2 Algorithm Configuration

This section is concerned with the algorithm's configuration during the experimental runs. In its first subsection, it will detail the parameter settings used as standard configuration of the examined algorithms. Based on these reference configurations, chapters 5 through 7 will examine the impact of specific parameter changes. Subsection 4.2.2 explains the termination criteria that were used for different test instances and shows in which cases local search has been applied. To further detail the latter, the third subsection provides some information on the chosen local search method.

4.2.1 Parameter Reference Configurations

For reasons of a structured comparison, a reference configuration has been defined for all examined algorithms, i.e., for ACS, MMAS, MMASq_0, and RANKBAS. The basic AS algorithm will not be examined any further as the named methods promise clearly superior performance [22]. When changing specific parameters in subsequent chapters, all other parameters will take their values as per the reference configuration. Only parameters that are explicitly mentioned when proposing a new parameter setting technique may deviate from it.

Table 4.2 gives an overview of all reference configurations. For each algorithm, it assigns values to all subsequently adapted parameters, distinguishing configurations without and with local search. Further parameters exist, but where kept at a fixed value for all algorithm variants. One of these is α (i.e., the relative influence of the pheromone trail during tour construction), which takes the value 1.0 for all algorithms. The length of the candidate lists (i.e., *cand*) is generally set to 20. For both MMAS variants, λ is set to 0.05 and $f_{\lambda}^{restart}$ is 2.00002. Restarts take place no earlier than $iter_{min}^{stag} = 250$ iterations after the last improvement of the best solution with the respective checks on f_{λ}^{avg} taking place every 100 iterations.

It should be noted that the values presented in table 4.2 largely correspond to those proposed by Dorigo and Stützle [22]. The settings for MMASq_0 are the same as those for MMAS besides the additional parameter q_0. The latter is set in the same manner as in ACS. As the referenced book does not provide any recommendation for RANKBAS

parameter	ACS		MMAS		MMASq_0		RANKBAS	
	$s \in S_{\overline{ls}}$	$s \in S_{ls}$	$s \in S_{\overline{ls}}$	$s \in S_{ls}$	$s \in S_{\overline{ls}}$	$s \in S_{ls}$	$s \in S_{\overline{ls}}$	$s \in S_{ls}$
m	10	10	n	25	n	25	n	25
β	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
ρ	0.1	0.1	0.02	0.2	0.02	0.2	0.1	0.1
q_0	0.9	0.98	(0.0)	(0.0)	0.9	0.98	-	-
ξ	0.1	0.1	-	-	-	-	-	-
g_τ	-	-	2.0	2.0	2.0	2.0	-	-
w	-	-	-	-	-	-	6	6

Table 4.2: Parameter reference configurations

with local search, the settings for the case without local search were adopted and the number of ants was fixed similar to MMAS. The latter appeared reasonable as without local search, MMAS and RANKBAS also use the same scaling for the population size (i.e., $m = n$). Parameter flexibility has been limited compared to what was proposed by Dorigo and Stützle with respect to the setting of the parameter β for configurations without local search. While the mentioned authors propose a range from 2 to 5, a fixed value of 2 was used in this work. On the one hand, the value needed to be fixed in order to define one unique reference configuration. On the other hand, all other values from the respective range could have been chosen instead. One may argue that choosing the same beta for both cases (i.e., without and with local search) would increase the degree to which conclusions can be derived on the effect of using local search.

4.2.2 Instance-Specific Configuration

Independent from the algorithm configurations introduced in the previous subsection, some settings are chosen depending on the size of the used problem instance. They are described in this subsection. Table 4.3 summarizes the termination criteria used for specific problem sizes and whether or not local search is applied.

instance size (n) in cities	runtime limit in seconds	local search
100-400	300	-
800	1,200	-
1,500-3,000	1,000	2-opt
6,000	10,000	2-opt

Table 4.3: Runtime limits and local search configuration for different problem sizes

The termination criterion was generally given in the form of a runtime limit. As local search method, 2–opt has been chosen. It is explained in more detail in the next subsection. The small instances with 100 to 800 cities were generally examined without local search. Runs on larger instances were always executed with local search. From comparing the results for these two sets of instances, conclusions on the impact of local search on parameter sensitivity may be derived for the given ACO context.

4.2.3 Local Search

The local search method that has been used in the present work is *2-opt*, as proposed by Croes [13] in 1958. As many other local improvement methods, it takes some initial valid solution and tries to improve it by applying relatively small changes.

The general idea of 2-opt is to replace pairs of edges contained in the given tour by alternative edges if this exchange causes the overall tour length to decrease. This process continues until no more pairwise swaps exist that would improve the solution quality. The resulting tour is considered *2-optimal*.

For a more detailed examination of 2-opt, the reader is referred to work by Hoos and Stützle [45]. Further speedup techniques that were applied in the present work are also described therein. The used implementation applies techniques commonly referred to as *fixed radius search*, *candidate lists*, and *don't look bits*. The candidate lists used by the local search method were of length 40.

The method 2-opt has been chosen for two reasons. Firstly, it allows to demonstrate the effect of parameter changes with reasonable clarity. While there are other local search methods that are more powerful than 2-opt, this additional performance typically comes at a price. Using a stronger local search method, one would have to execute the experiments on larger problem instances and with a longer runtime to be able to observe the same behavior as with 2-opt.

Secondly, the relative weakness of the method has been considered a realistic assumption with respect to problems other than the TSP. I.e., in other contexts, the choice of available local search methods may be limited to fewer and oftentimes weaker alternatives. Accordingly, the use of 2-opt should increase the degree to which the subsequently presented results can be transferred to different optimization problems.

4.3 Software and Hardware

This section is intended to give the reader some general information on the experimental setup. The implementation used in the present work is based on the ANSI–C code by Thomas Stützle, as available online[1]. The code's core ACO functionality has not been subject to major changes unless subsequently noted. Nevertheless, the parameter setting methods described in the remainder of the work required to extend it in various regards.

The necessary runs for the experimental analysis were executed on a cluster of Intel Xeon E5410 quad core CPUs (4 x 2.33 GHz, 6MB L2 cache). One computational node of the cluster contained two of these CPUs, which shared a total of 8 GB main memory. The experiments were run in parallel and independent from one another. Each algorithm run was executed on exactly one CPU core with an average of roughly 1 GB main memory available. It has been verified that the code did not exceed the available memory (i.e., no swapping effects need to be taken into account). The cluster ran the Linux distribution *Rocks* 4.2.1 with underlying *CentOS* 4.3.

[1]http://www.aco-metaheuristic.org/

5 Static Parameter Settings

This chapter presents the analysis that has been undertaken in the scope of this work regarding fixed parameter settings. By varying the previously introduced reference configurations with respect to one parameter at a time, the algorithm's sensitivity to changes in the considered parameters was investigated. The goal of this analysis was to identify parameters with potential for improving algorithm performance when being adapted in the course of a run.

After reviewing the examined algorithm configurations in the first section, the respective results for MAX-MIN Ant System will be presented in section 5.2. The chapter closes with a third section performing some analysis on the different algorithms' reference configurations.

5.1 Examined Configurations

Table 5.1 in combination with table 5.2 gives an overview of the experiments that were run in the context of this chapter. All experiments used the algorithm configurations as introduced in section 4.2 as reasonably well performing benchmark setups. Starting from there, a series of runs were executed, each varying one parameter at a time. Table 5.1 shows the parameters that have been varied as well as the specific values that were used for each of them.

m	β	q_0	ρ	ξ	g_τ	w
1	0.01	0.00	0.01	0.01	0.25	1
2	0.05	0.01	0.02	0.02	0.50	2
3	0.10	0.05	0.05	0.05	1.00	4
5	0.25	0.10	0.10	0.10	2.00	6
7	0.50	0.25	0.20	0.20	3.00	8
10	0.75	0.50	0.40	0.40	5.00	10
15	1.00	0.75	0.60	0.60	10.00	15
25	1.50	0.90	0.80	0.80	50.00	20
50	2.00	0.95	0.90	0.90		30
100	3.00	0.99	0.95	0.95		50
200	4.00	1.00	1.00	1.00		
400	5.00					
800	10.00					
1,600	20.00					

Table 5.1: Fixed parameter settings as used in the trade-off analysis

The sets of test points as shown in table 5.1 have been chosen to provide a representative spread across the value ranges that were considered reasonable for the respective parameters. Typically, more test values were chosen towards the extreme ends of the value ranges, especially the lower end. This was done because small changes in these regions were expected to have a larger relative effect on algorithm behavior.

With respect to the number of ants (i.e., m) it needs to be noted that not all instances were examined with all population sizes in the list. On runs with local search, the highest number of ants that was considered is 200. Without local search, all population sizes up to $2 \cdot n$ ants were examined. The latter accounts for the increasing upper limit suggested by the recommended setting of $m = n$ as introduced in subsection 4.2.1.

Table 5.2 summarizes, which parameters where adapted on which of the algorithms introduced in chapter 2. The table reflects all experiments with static parameter settings that were executed in the context of this thesis. However, due to space constraints, not all results can be presented in this document. In the following, only the results for MMAS will be detailed further. This choice has been made as the deterministic, respectively adaptive parameter control methods proposed in chapters 6 and 7 were both developed for MAX-MIN Ant System. Respective result graphs for ACS, MMASq_0, and RANKBAS have been generated and the author will provide them to the interested reader.

	m	β	q_0	ρ	ξ	g_τ	w
ACS	x	x	x	x	x	-	-
MMAS	**x**	**x**	**x***	**x**	**-**	$s \in S_{ls}$	**-**
MMASq_0	x	x	x*	x	-	-	-
RANKBAS	x	x	-	x	-	-	x

Table 5.2: Overview of parameters that were examined for specific algorithms

The two combinations marked with x^* in table 5.2 are effectively the same as MMAS and MMASq_0 differ only in their standard configuration's value for q_0. One should also note that the effect of changing g_τ is examined only for cases with local search, as the parameter is used only in this setting (see equations 2.22 and 2.23).

Furthermore, the reader should be aware that the one-dimensional test setup described above is not without limitations. Changing only one parameter based on a fixed reference configuration is clearly inferior to a full factorial test design considering all possible combinations of parameter values. While the latter is potentially able to identify various local optima in the parameter space, the one-dimensional approach essentially examines the local neighborhood of one specific parameter setting. Choosing this setting according to widely accepted recommendations ensures to some degree that this search for improvements is centered on a reasonable region of the parameter space.

Given the number of algorithms and parameters examined, a full factorial test layout would require the availability of enormous computational resources, by far exceeding the considerable number of CPU years underlying the present study. To examine only a single problem instance using MMAS with local search, one would have to examine $149,072$ possible configurations. Taking further instances into account and expecting to run each configuration several times to obtain a somewhat reliable measure of average performance illustrates the impracticability of this approach. Thus, should similar studies be conducted in the future, they need to be carefully designed. To get within realistic ranges of required CPU time, the study needs to be focused on a rather small set of reasonably discretized parameters. One may also want to consider using an iterative refinement strategy. In the latter case, one would start out by examining a limited set of parameter settings that are in some way equally distributed across different regions of the parameter space. In later refinement steps, one could explore the promising regions in more detail.

5.2 Results for MAX-MIN Ant System

Before subsequently discussing the obtained results, a couple of general aspects about selection and format of the former should be noted. The reader may, e.g., use the plots shown in figure 5.1 to illustrate the aspects mentioned below.

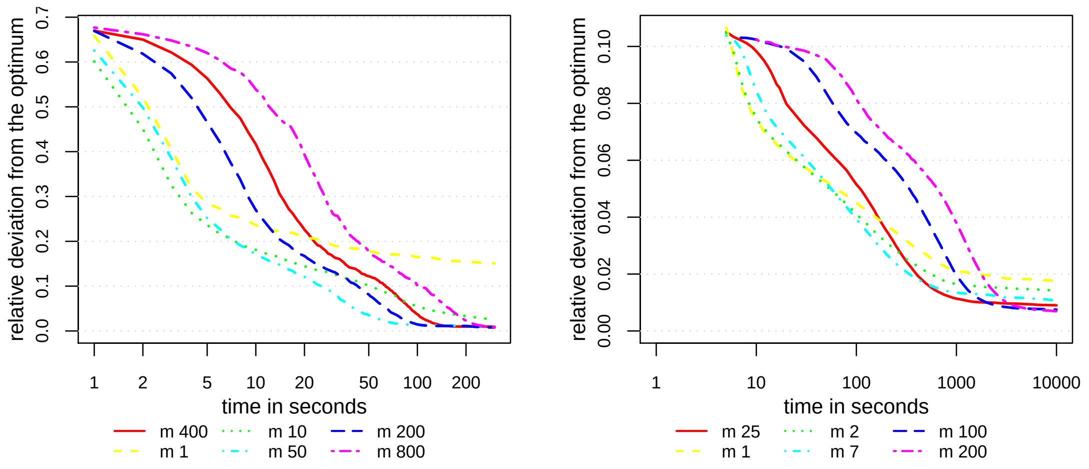
After initial considerations of examining algorithm performance only at the end of the runtime, it was decided to examine the performance of different parameter settings along the whole algorithm run. This allows to better contrast the trade-offs incurred between different parameter settings. While some may perform exceptionally well in early stages of the run, they may be inferior to others later on. Observing such behavior may give suggestions on how a parameter's value should change in the course of an algorithm run to improve performance.

All shown performance curves depict the average deviation from the optimum across 25 algorithm executions with the same parameter setting. In general, all curves presented in this thesis, even if depicting other measures, are averaged over 25 runs.

It is important to note that the x-axis gives the algorithm runtime using a logarithmic scale. When a plot shows no data points in the first seconds, as in figure 5.1(b), this can be attributed to the algorithm's initialization phase. The length of the latter is positively related to the size of the considered problem instance.

The first curve to be shown in subsequent performance plots always corresponds to the algorithm's reference configuration. It is typically a continuous red, respectively dark (on b/w prints) line. To improve the legibility of the plots, each of them shows a maximum of six curves. These capture the main trade-offs regarding the values in table 5.1.

From the examined set of problem instances (see table 4.1), the instances 400-1 and 6000-1 have been selected as representative examples for cases without, respectively with local search. While the idea was to generally select the largest possible representative of each category, the instances with 800 cities were discarded as their $1,200$ seconds runtime typically cut off the last stage of the convergence process. For algorithm configurations that were examined only with local search, the instances 1500-1 and 6000-1 will be shown in the plots. Again, the author provides the remaining results upon request.



(a) instance 400-1: varying m, no local search (b) instance 6000-1: varying m, 2-opt

Figure 5.1: Performance plots for varying m in MMAS

5.2.1 Varying the Number of Ants

Starting with the number of ants as the first parameter to be varied, figure 5.1 provides the respective performance plots. Both plots generally display the same trends. For many population sizes, a near parallel development of solution quality can be observed. The plots also indicate that small population sizes perform well for relatively short runtimes, while larger populations perform better at later stages of the run.

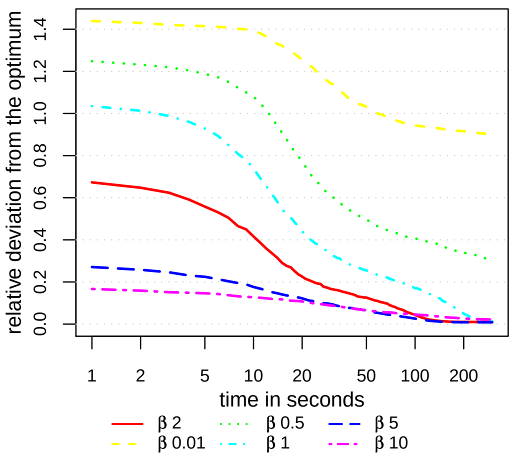
For the case with local search, the second observation holds for all examined population sizes, i.e., larger population sizes seem to always improve over smaller ones if runtime permits. This is especially interesting because it gives rise to ideas for potential parameter setting schemes, as investigated in the next chapter. E.g., it suggests that it may be a good idea to start with a small population, which then increases in the course of the run.

Without local search, only rather small population sizes are inferior to larger ones. Figure 5.1(b) shows several population sizes significantly smaller than the problem size n (i.e., the reference value) that are able to outperform the latter in early stages without paying a price in terms of final solution quality. This indicates that the recommendation of setting $m = n$ by Dorigo and Stützle [22], which has been derived using rather small TSP instances, does not hold for larger ones.

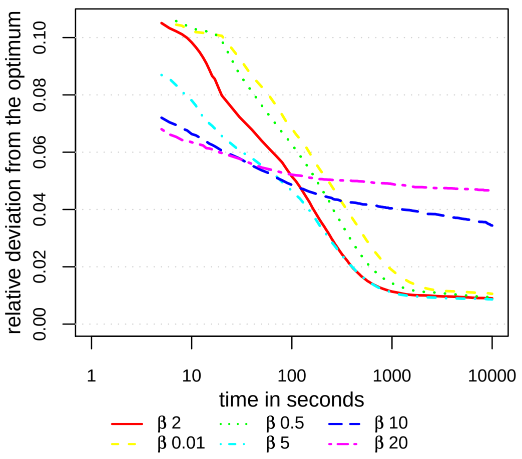
Another aspect that should not go unnoticed is the different order of magnitude with respect to solution quality that goes along with the application of local search. While the small instance without local search starts at close to 70% deviation from the optimum (i.e., the average best known solution at the end of the first second), the application of 2-opt on a significantly larger instance reduces this initial gap to slightly more than 10%.

5.2.2 Varying the Impact of the Heuristic

The next parameter that has been subject to investigation is β. Figure 5.2 shows performance curves obtained by running MMAS with different values for β. The first thing to be noted is that high values of β apparently start off at significantly better solutions than lower ones. This can be attributed to the fact that the higher the value of β, the closer the constructed tours are to a nearest neighbor tour. As it takes some time for the pheromone



(a) instance 400-1: varying β, no local search



(b) instance 6000-1: varying β, 2-opt

Figure 5.2: Performance plots for varying β in MMAS

trails to actually evolve exploitable knowledge, runs with low β values spend more time on a rather random exploration of the search space before they finally start to converge.

A negative effect of using a rather high β is that the knowledge that gets evolved over time can barely impact the tour construction process. This is the reason why too high β values make it hard for the algorithm to further improve their good initial solutions. This is clearly visible for the case with local search when considering the curves for $\beta = 10$ and $\beta = 20$. But also without local search, runs with β values of 10 or more display a weak performance towards the end of the run. Nevertheless, it is these high values which in the latter case prove superior too many other configurations for significant shares of the algorithm's runtime.

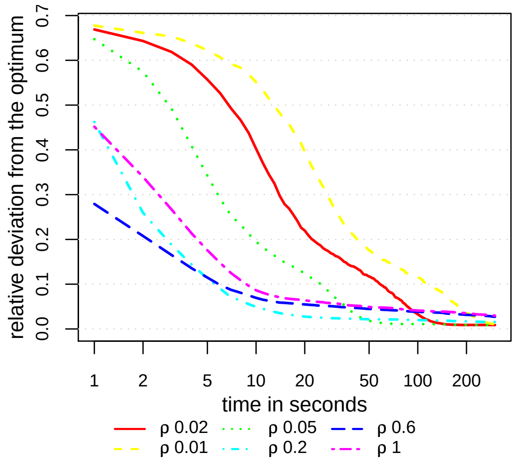
Again, the above observations can be used to derive an intuition for how the parameter β should be changed in the course of an algorithm run. One could, e.g., start with a high value for β and decrease it subsequently.

Different from the previously examined parameter m, the algorithm's performance seems to be a lot more sensitive to the value of β. While in figure 5.1 most curves start and end within the same region of solution quality, figure 5.2 shows rather large variations - primarily for the case without local search. One may be able to argue that the application of local search seems to pull the curves corresponding to reasonably-sized βs closer together.

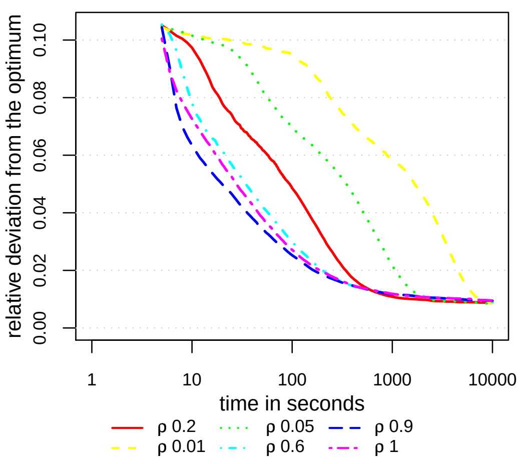
To some degree, the results also confirm the recommendations by Dorigo and Stützle [22] which mention the fixed value 2 and the interval $[2, 5]$ as well-performing settings for runs without and with local search, respectively. In fact, a fixed β of 5 seems to perform best for both configurations. In future extensions of this work, one may want to consider modifying the reference configuration accordingly.

5.2.3 Varying the Evaporation Rate

The third considered parameter is ρ, whose performance is shown in figure 5.3. Without local search, medium-sized values seem to perform best in the beginning, while being inferior to smaller ones as the algorithm converges. In terms of potential adaptation scenarios, this suggests that reproducing a similar behavior of ρ (i.e., decreasing it from



(a) instance 400-1: varying ρ, no local search



(b) instance 6000-1: varying ρ, 2-opt

Figure 5.3: Performance plots for varying ρ in MMAS

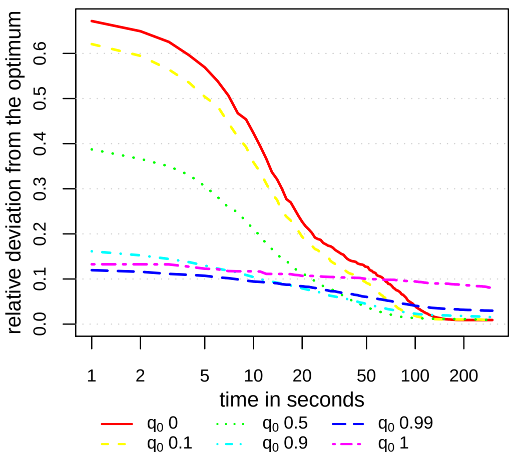
mid-sized to low values) during the run could increase algorithm performance. With local search, similar trends persist with high evaporation rates playing a more important role in the beginning. Towards the end, they are again somewhat worse than smaller ones, but with no clear advantage of real small evaporation rates (e.g., $1\% - 5\%$) over moderate ones (e.g., 20%). Accordingly, it may be a reasonable idea to decrease ρ rather fast from a high to a low level.

Again, the values recommended by literature perform well with respect to the solution quality at the end of the run. For the cases without local search, the results suggest that using $\rho = 0.05$ may reach the same level of final solution quality as the standard setting while performing better for significant parts of the run.

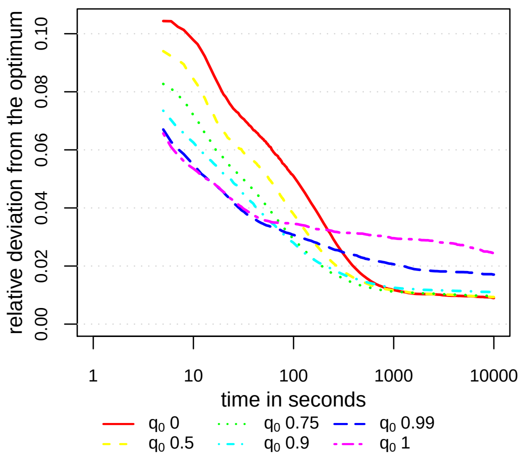
Taking the actual meaning of the parameter ρ into account, the results presented in figure 5.3 also show a rather surprising detail. Choosing the evaporation rate as 1.0 essentially corresponds to switching off the long-term memory of the ant algorithm. The amount of newly deposited pheromone (see equation 2.19) would still survive to the next iteration, but all previously deposited pheromone would inevitably be lost. Effectively, this transforms the ACO algorithm into something similar to a stochastic local search procedure. The fact that the latter performs quite well on a multitude of problems may explain the surprisingly good results for high values of ρ. If the MAX-MIN Ant System with $\rho = 1.0$ however turned out to strictly outperform configurations with typical (i.e., rather low) evaporation rates, this would seriously question the benefit of using stigmergy in this context. In fact, both, in cases without and with local search, the configurations using a small evaporation rate are able to perform better than this extreme setting with respect to their final solution quality.

5.2.4 Varying the Tour Construction's Elitism

Figure 5.4 provides the performance plots of the next examined parameter. Changing q_0 is somewhat untypical for MAX-MIN Ant System, as in its standard setup, the algorithm does not have this parameter. The required changes have been introduced at the end of section 2.3 in the context of MMASq_0. The latter alias is generally used in the present



(a) instance 400-1: varying q_0, no local search



(b) instance 6000-1: varying q_0, 2-opt

Figure 5.4: Performance plots for varying q_0 in MMAS

work to refer to an MMAS variant using a fixed positive q_0 in its reference configuration (see table 4.2). Table 5.2 previously indicated that when varying the value of q_0, the experimental setup is the same for MMAS and MMASq_0 (i.e., both use pseudorandom proportional selection as opposed to standard MMAS).

Generally, it can be observed that a high q_0 causes the algorithm to start off at better initial solutions. Similar to starting at a high β, it causes the algorithm to mostly search around nearest neighbor tours in early stages of a run. While the former reaches this by explicitly focusing on the heuristic values, the latter does it rather implicitly by increasing the degree of elitism in tour construction. As the pheromone levels are all rather equal in the beginning, the major factor influencing tour construction is again the heuristic value. Thus, a more elitist tour construction strengthens the impact of the heuristic values in early stages of the run.

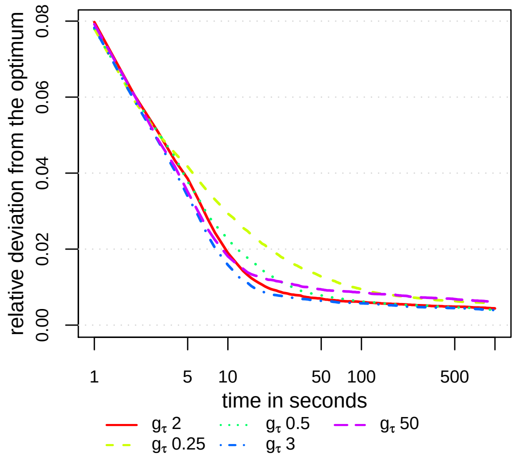
While a high q_0 does not diminish the effect of different pheromone levels (as, e.g., a high β), it nevertheless limits the algorithm's exploration behavior by avoiding locally sub-optimal choices during tour construction. Accordingly, rather small values of q_0 seem to perform best with respect to the final solution quality, whereas larger values appear to have problems with reaching equivalent performance levels.

Similar to the previous results, the above observations give rise to a suggestion on how the parameter q_0 could possibly be adapted. It appears to be worth investigating to start with a high value, which is consecutively decreased to a low one, possibly to zero.

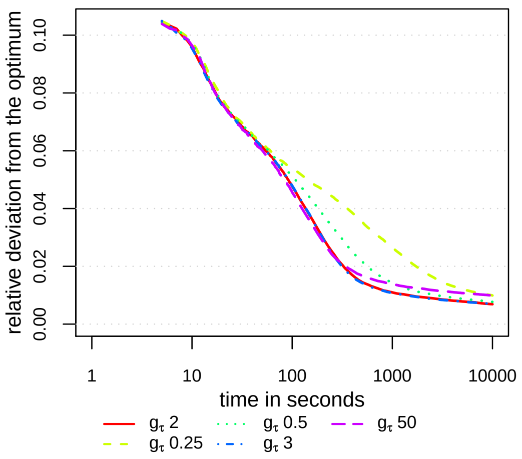
The plots clearly show that standard MMAS (i.e., $q_0 = 0$) is generally among the best performing configurations with respect to the solution quality reached at the end of a run. Nevertheless, choosing a reasonably small positive q_0 may be competitive towards the end, while performing better than the reference configuration in early stages of the run.

5.2.5 Varying the Minimal Pheromone Level

For MMAS runs with local search, one more parameter has been subject to experimental analysis. The parameter g_τ effectively depicts the value of the minimal pheromone level relative to the maximal one (see equation 2.22). The algorithm behavior for different values



(a) instance 1500-1: varying g_τ, 2-opt



(b) instance 6000-1: varying g_τ, 2-opt

Figure 5.5: Performance plots for varying g_τ in MMAS with local search

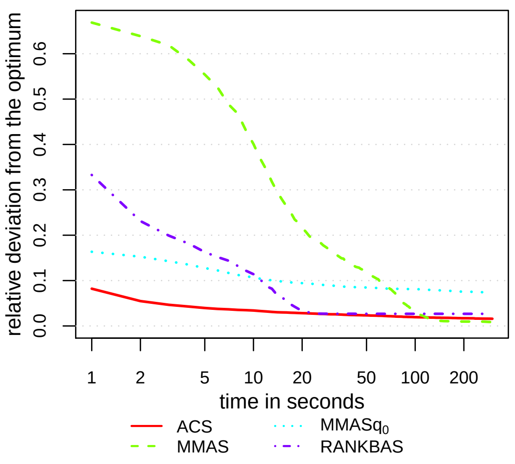
of g_τ is depicted in figure 5.5. As the respective runs for this pair of graphs were executed in the context of experiments for chapter 7, a slightly different algorithm configuration has been used here. As opposed to all previous results, the latter were obtained with MMAS pheromone re-initializations disabled (i.e., $iter_{min}^{stag} = \infty$).

It is important to note that both graphs correspond to runs with local search, one on the instance 1500-1, the other on 6000-1. Both graphs show only little sensitivity with respect to the setting of g_τ. Extremely small as well as extremely large values for the parameter perform worse than the reference configuration. Even though there is no clear advantage for the 6,000 city problem, the 1,500 city instance suggests that $g_\tau = 3$ may perform slightly better than $g_\tau = 2$ (i.e., the standard). In either case, the curves do not show enough variation, respectively no clear sequence of best parameter values that could give rise to ideas for promising parameter setting schemes. Accordingly, the idea of adapting g_τ during an algorithm run has not been investigated further at this point.

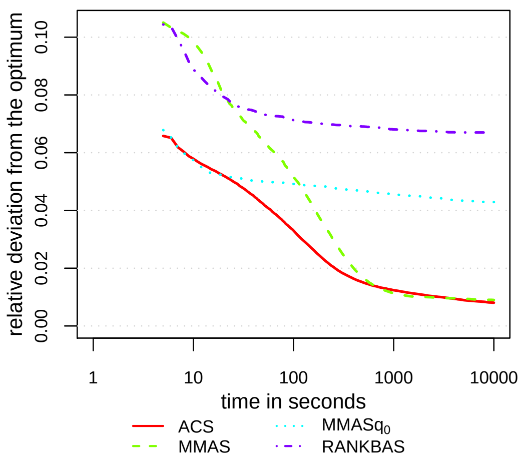
5.2.6 Summary

In summary, this section was able to illustrate the trade-off between different settings for the parameters m, β, ρ, q_0, and g_τ. For many of these, it could be observed that certain configurations perform better in early stages of the run while others perform best at a later point. This has led to the development of first ideas on how the respective parameters would have to change in the course of a run in order to optimize algorithm performance across different running times. The development of the parameter setting methods presented in the next two chapters has been guided by this intuition.

The author believes that future analysis on fixed parameter settings in the given context should in a first step aim at identifying further local optima in the parameter space. This could be done by algorithmic means such as the previously introduced iterated F-Race (see subsection 3.2.3) or, if time permits, one could examine some kind of a full factorial test layout. The results of such an analysis could allow to propose an improved reference configuration and could provide intuition, which parameters one should change in parallel in order to move between different well performing regions in the parameter space.



(a) instance 400-1: no local search



(b) instance 6000-1: 2-opt

Figure 5.6: Performance plots for the different algorithms' reference configurations

5.3 Taking a Glance at the Reference Configurations

This section is divided into two rather short parts. The first subsection contrasts the performance of the different Ant System variants' reference configurations. For MMAS, the second subsection sketches the relationship between algorithm iterations and runtime, thereby providing insights that will prove helpful in the context of chapters 6 and 7.

5.3.1 Performance Comparison

This section is intended to give the reader a brief overview of the performance levels on which the other examined algorithms operate compared to MMAS. Figure 5.6 shows the respective plots.

Especially at the beginning of the run, the curves for ACS show large advantages in solution quality over all other AS variants. The reference configuration of MMAS as considered in this work is strongly outperformed in early stages of the run. However, it is also the one catching up most as the runtime increases. Both, for runs without and with local search, the smaller ones among the examined instances show MMAS improving over ACS towards the end. For the largest examined problem size (i.e., 6000 cities), however, ACS seems to take back its lead after having been inferior to MMAS for a short while.

The initial difference between ACS and MMAS can be explained. The latter method initializes the pheromone trails at their maximum value τ_{max} whereas ACS uses its implicit lower bound τ_0. This initialization keeps the MMAS pheromone levels relatively homogeneous until enough pheromone is evaporated. In contrast to this, the ants in ACS create significant differences among the pheromone trails right from the first generation. In addition, the lower population sizes in ACS allow to execute a significantly higher number of iterations than in MMAS. This may be a reason for ACS finding good solutions quite early, whereas MMAS spends a significant amount of time exploring larger regions of the search space in a less elitist manner. Further supportive reasoning on the effect of using the pseudorandom proportional selection has been given in the previous section.

Compared to the previous two algorithms, MMASq_0 performs quite poor. Its initial solution quality is able to strongly profit from the algorithm using the abovementioned selection method during its tour construction. However, as shown when examining the effect of q_0 on MMAS in the previous section, the high elitism implied by large fixed q_0 values comes at the cost of poor convergence behavior in later stages of the run.

Considering Rank-based Ant System, it needs to be stated that this method performs rather well in configurations without local search. Once the algorithm reaches a reasonably good level of solution quality, many of the obtained results however seemed to show a stagnation of the search. With local search, it needs to be noted that the reference configuration as proposed in section 4.2.1 performs extremely poor and cannot be considered a general recommendation in this context. If one were able to break the stagnating behavior of RANKBAS, further work on identifying good parameter settings with local search may yield reasonable results in the future.

5.3.2 Relating Iterations to Time for MAX-MIN Ant System

The two next chapters deal with parameter setting approaches that in some way use the value of the iteration counter to determine when or how much to change a parameter's value. Thus, when trying to put these methods into context with time-based performance evaluations, it is important to understand how time and iterations relate to one another.

Table 5.3 shows an approximate mapping of time to algorithm iterations for a selected set of problem sizes. These were derived using randomly chosen runs with the MMAS reference configuration. The reader needs to note that the depicted mapping is not intended to serve as an exact translation and that it does not apply to arbitrary algorithm configurations. Nevertheless, the shown values can be used as a good indicator of the relevant order of magnitude when subsequently trying to relate time to algorithm iterations.

time in	approximate number of iterations				
seconds	$n = 400$	$n = 800$	$n = 1500$	$n = 3000$	$n = 6000$
5	66	12	86	18	0
10	126	25	201	41	7
20	254	51	470	110	20
50	620	119	1,260	337	70
100	1,260	240	2,600	765	168
200	2,620	490	5,200	1,639	430
500		1,209	13,050	4,144	1,270
1,000		2,467	26,760	8,800	2,725
2,000					5,460
5,000					16,000
10,000					32,000

Table 5.3: Approximate relationship of seconds of runtime to executed algorithm iterations for the MMAS reference configuration; selected problem sizes

6 Deterministic Parameter Schedules for MAX-MIN Ant System

Based on the findings from the previous chapter, chapter 6 will propose a series of deterministic schemes to control different MMAS parameters. The adaptation of the parameters is performed at the beginning of every iteration of the algorithm's main loop. With respect to the pseudo code presented on page 5, the parameter adaptation would be inserted right in front of the ants' tour construction.

Several types of parameter schedules were examined. Many of them are inspired by observations from chapter 5. Others were created out of curiosity to see whether changing a parameter in a less intuitive manner may benefit the algorithm. Below, the general idea of the different adaptation schemes is explained.

The first two schedule types change parameters in a linear, respectively exponential manner. They essentially aim at tracking a good value of the adapted parameter throughout the run. *Linear schedules* increase or decrease the respective parameter by adding or subtracting a particular amount at regular intervals. One can formalize the logic underlying a linear schedule for a parameter p as

$$p = \left| p_0 + p_1 \cdot \left\lfloor \frac{iter}{k} \right\rfloor \right|_{lbound}^{ubound} \tag{6.1}$$

where $|x|_{lbound}^{ubound}$ symbolizes that the parameter is lower or upper bounded by some value. In principle, the value of the parameter is computed from a starting value p_0 that is adjusted by an amount p_1 every k iterations. Whenever the adaptation crosses the upper or lower bound, the parameter's value is fixed to the respective bound and the adaptation ends. *Exponential schedules* work quite similar. They exponentially increase or decrease the respective parameter by multiplying its value with a fixed percentage every k iterations. An equivalent mathematical expression can be easily derived by replacing the summation in equation 6.1 by a multiplication.

The idea underlying *one-step schemes* is somewhat different. They are based on the notion of one particular parameter value performing well in early stages of the run and another fixed value being good for the remaining runtime. Typically, the latter of the two is chosen with a focus on the level of solution quality reached towards the end of the run. In general terms, one-step schemes start with a fixed value for the considered parameter and set the latter to a different, fixed level after a certain number of iterations. In this case, the formalization is quite straightforward:

$$p = \begin{cases} p_0 & \text{if } iter \leq k \\ p_1 & \text{else} \end{cases} \tag{6.2}$$

In iteration $k+1$, the value of parameter p switches from p_0 to p_1. Similar expressions can be derived to formalize the subsequently explained schedule types, but are omitted at this point for the sake of simplicity.

Iterative schemes alternate between different parameter levels. E.g., one could use a specific parameter level for k_1 iterations and then use a different value for the next k_2 iterations. This sequence would be repeated throughout the run. This type of parameter schedule is derived from the fuzzy notion of regularly switching back and forth between parameter values that foster exploration, respectively exploitation.

Similarly, *sinus schemes* try to give the algorithm the chance to use all relevant parameter levels on a regular basis. They set the respective parameter by means of a sinus curve, which is positioned (i.e., shifted and scaled) upon a particular subrange of the considered parameter's permitted value range. The curve repeats itself every k iterations.

Random schemes set the respective parameter randomly. Depending on the context, different distributions may be used. The examination of these schemes primarily aims at determining whether it is possible to introduce something like a useful noise into the search process. The latter is based on the idea of preventing search stagnation without harming the algorithm's ability to converge to good quality solutions. Some of the used random schemes also have aspects of linear adaptation. I.e., some schemes using a uniform random distribution change the distribution's value range in a linear manner as the search progresses. Labeling a random method as deterministic parameter control is rather unintuitive. However, this is correct in terms of the classification scheme introduced in section 3.1. It is the method's property to choose the parameter values without taking any aspects of the algorithm behavior into account that makes it deterministic.

With respect to the analysis performed in the previous chapter, the parameters m, β, ρ, and q_0 showed some potential for deterministic adaptation. They will thus be examined further in the following.

Again, only a subset of the obtained results will be presented in detail, i.e., only selected parameter control schedules will be depicted in the plots. The methods were primarily selected based on their performance. Details on how the shown schemes work will be provided along with the results. Upon request, the author provides further information on properties and performance of the remaining schedules that have been examined.

One important aspect to generally be noted about the subsequently presented results is the following: Whenever MMAS executes a pheromone re-initialization, the parameter schedules are also restarted. Effectively, the respective parameter is reset to its initial value and the adaptation starts from scratch. Whenever the iteration counter *iter* plays a role in setting a variable, the number of iterations since the last restart will be used as the relevant measure. A restart of the adaptation schedule was considered reasonable as a re-initialization of the pheromone values creates a situation that is in many regards similar to the one at the beginning of the run. Thus, similar parameter settings should be applied.

6.1 Adapting the Number of Ants

The number of ants is the first parameter to be examined here. The results presented in section 5.2 indicated that increasing the number of ants during a run may be a promising approach (see figure 5.1). While small population sizes perform well early on, larger ones guarantee the better solution quality towards the end of the run.

The subsequently shown list provides details with respect to the schedules evaluated in figure 6.1. Each schedule has an identifier of the form $a_x\, i$ where x is the adapted parameter and i is an index distinguishing different schedules for the same parameter. When a curve is labeled $x\, j$, this denotes that the curve uses a fixed parameter setting of x equal to j.

a_m 1: linear schedule, starting at $m = 1$ and adding 1 ant every iteration

a_m 2: linear schedule, starting at $m = 1$ and adding 2 ants every iteration

a_m 3: linear schedule, starting at $m = 1$ and adding 5 ants every iteration

a_m 4: linear schedule, starting at $m = 1$ and adding 1 ant every 2 iterations

a_m 5: linear schedule, starting at $m = 1$ and adding 1 ant every 5 iterations

a_m 6: linear schedule, starting at $m = 1$ and adding 1 ant every 10 iterations

a_m 14: iterative scheme, alternating between 15 iterations at $m = \lfloor iter/64 \rfloor$ and 5 iterations at $m = \lfloor iter/16 \rfloor$

a_m 18: random scheme setting m according to a uniform random distribution on the interval $[1, \lfloor iter/16 \rfloor]$

While adaptation schemes a_m 14 and a_m 18 are of rather exotic nature, the schedules with indexes 1 through 6 simply increase the number of ants in a straightforward manner. By comparing the different linear schedules, one can derive conclusions on which speed of the increase to choose.

Subfigures 6.1(a) and 6.1(b) show three different well-performing schedules compared to three fixed population sizes for cases without, respectively with local search. The fixed population sizes were chosen as the standard setting and two curves intended to depict the best possible performance at different stages of the run.

The plots show that the deterministic schedules are able to improve the performance of the reference configuration in early stages of the run without any loss in solution quality in later stages. Furthermore, the respective performance curves are able to track or improve the lines corresponding to the best-performing runs with fixed population sizes. It is especially interesting that for configurations with local search, the schedules are even able to improve over the reference configuration with regard to their final solution quality. They reach the performance level of large fixed population sizes (see curve for $m = 200$ in figure 6.1(b)), while at the same time performing as good as significantly smaller populations in early stages of the run.

Two interesting observations can be made with respect to the performance of the random population sizing schemes that were examined. While they perform similarly well as linear schemes if no local search is applied, one of them shows a surprisingly good performance in the case with local search (see figure 6.1(b)). For several hundred seconds, a_m 18 performs better than all other configurations, even better than the best tested fixed population sizes. To some degree, this shows that setting the population size in a random manner can be more than just a weird way to reproduce a straightforward methodology's behavior.

While more experimentation is needed to explain this particular situation, another interesting observation can be made when using random schemes working on a fixed interval (i.e., the distribution's value range does not change based on the iteration counter). It turns out that the behavior of some random schemes can actually be related to configurations using a fixed population size. E.g., runs using a fixed number of 400 ants show exactly the same behavior as choosing the population size using a uniform random distribution between 1 and 800 (i.e., the expected value is about 400). Due to the rather simple nature of the respective plots, they are not shown in this document. It appears that in this case, the performance is dependent on the average number of ants available to the algorithm, rather than the exact population size in a particular iteration. However, it is hard to argue along these lines for random schemes that change the distribution's value range over time. With respect to figures 6.1(a) and 6.1(b), the average number of ants in iteration $iter$ is approximately $iter/10$ for $a_m\,6$, whereas it is $iter/37$ for $a_m\,14$ and $iter/32$ for $a_m\,18$. Despite the significantly different average population size of $a_m\,6$, all three methods show relatively similar performance.

Figures 6.1(c) and 6.1(d) contrast further linear population sizing schemes' performance. The plots show schemes that increase the number of ants at different speeds in order to illustrate the sensitivity of linear schemes with respect to this factor. Among the examined schedules, slow schedules generally perform better than faster ones during most stages of the run. It may be interesting to run experiments with even slower schedules in the future.

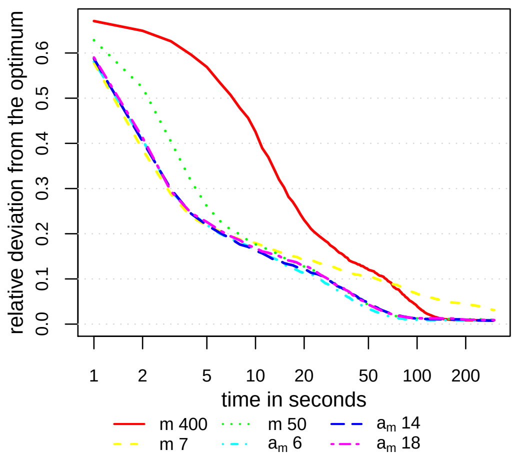
However, for configurations with local search, things look somewhat different when approaching the end of the run. In this case, schemes increasing the population size rather quickly are able to perform better than the reference configuration, rather than schemes working with fewer ants. It may be interesting to investigate in the future whether this is due to the larger number of ants used at the end of the run or whether the continuous increase in ants is a precondition for good convergence behavior in later stages.

Besides the above disadvantage, rather slow linear schemes like $a_m\,6$ consistently proved to be among the best-performing population sizing schedules, independent from the considered problem size. Thus, the question on how fast to increase the number of ants may be rather a question of the required solution quality and runtime constraints and not that much a matter of instance size.

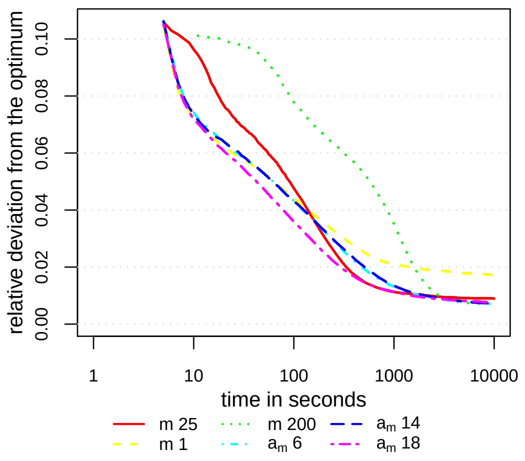
In summary, this section has shown that the concept of deterministic schedules for the population size is not only applicable, it is also able to improve the algorithm's performance throughout significant stages of the run. Especially with local search, using an increasing population size improves the quality of the final solution while still improving over the reference configuration in early stages.

6.2 Adapting the Impact of the Heuristic

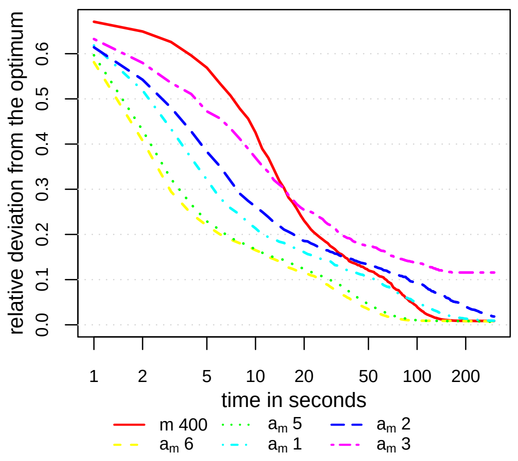
The parameter β, i.e., the relative importance of the heuristic information during tour construction, is the second parameter that showed potential for online adaptation in section



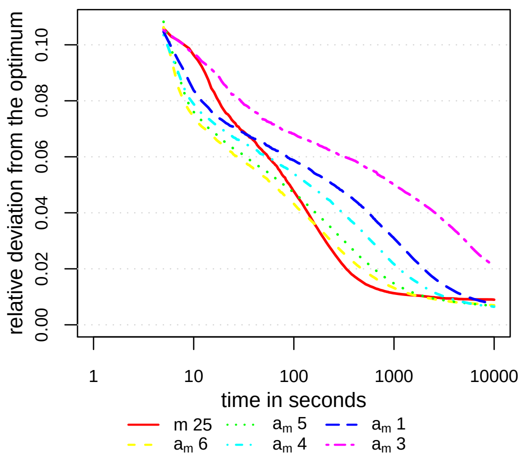
(a) instance 400-1: varying m, no local search



(b) instance 6000-1: varying m, 2-opt



(c) instance 400-1: varying m, no local search, focus schedule sensitivity



(d) instance 6000-1: varying m, 2-opt, focus schedule sensitivity

Figure 6.1: Performance plots using deterministic schedules to set m in MMAS

5.2. The results shown in figure 5.2 suggested that it may be a good idea to start with a quite high value of β and to subsequently decrease it to a reasonable level, e.g., to a value close to the one of the reference configuration. Figure 6.2 shows the results that were obtained using the following schedules:

a_β 33: linear schedule, starting at $\beta = 8.0$ and subtracting 1.0 every 50 iterations until $\beta = 3.0$

a_β 37: one-step scheme, starting at $\beta = 20.0$ and setting $\beta = 1.0$ after 50 iterations

a_β 38: one-step scheme, starting at $\beta = 20.0$ and setting $\beta = 2.0$ after 50 iterations

a_β 39: one-step scheme, starting at $\beta = 20.0$ and setting $\beta = 3.0$ after 50 iterations

a_β 40: one-step scheme, starting at $\beta = 20.0$ and setting $\beta = 1.0$ after 100 iterations

a_β 41: one-step scheme, starting at $\beta = 20.0$ and setting $\beta = 2.0$ after 100 iterations

a_β 42: one-step scheme, starting at $\beta = 20.0$ and setting $\beta = 3.0$ after 100 iterations

a_β 43: one-step scheme, starting at $\beta = 20.0$ and setting $\beta = 1.0$ after 250 iterations

a_β 44: one-step scheme, starting at $\beta = 20.0$ and setting $\beta = 2.0$ after 250 iterations

a_β 45: one-step scheme, starting at $\beta = 20.0$ and setting $\beta = 3.0$ after 250 iterations

Subfigures 6.2(a) and 6.2(b) show the best-performing β adaptation schedules compared to the reference configuration and two well performing fixed β settings. It becomes apparent that some β schemes significantly improve the initial solution quality, particularly for runs without local search. The one-step adaptation schemes are able to track the best curves using a fixed β at all times. The fact that the shown linear schedule $a_\beta\,33$ is slightly inferior in early stages of the run can be attributed to its lower starting value for β.

Comparing the schedules $a_\beta\,39$ and $a_\beta\,45$ for the case without local search suggests that using a high β for a longer period of time may slightly increase the solution quality obtained in intermediate stages. At the same time, this improvement seems to come at the cost of a slightly worse performance in later stages.

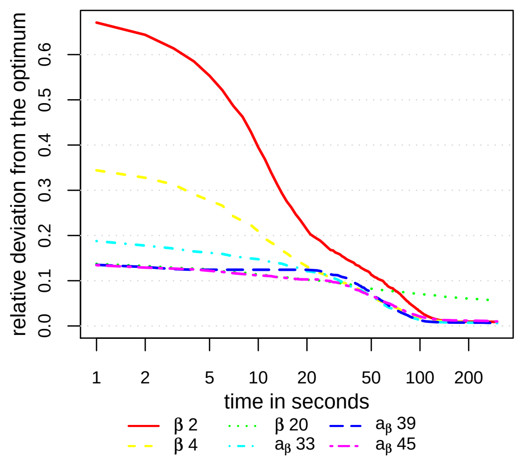
The two remaining plots in figure 6.2 examine how the behavior of one-step schemes changes, when varying the number of iterations after which the stepping occurs or when changing the value to which β is decreased.

Starting with the case without local search, several interesting observations can be made. The different lines in the respective plot all share a specific property: They evolve relatively flat until they seem to hit some *magic marker* and start to improve rapidly. Analyzing the obtained results at some detail, it became apparent that this magic marker actually closely corresponds to the performance curve using the adaptive scheme's target value as fixed setting for β. That means $a_\beta\,41$ is essentially not improving significantly until it hits the curve for $\beta = 2.0$, which is what the mentioned schedule steps down to. The same observation holds for $a_\beta\,40$ and $a_\beta\,42$ with the respective fixed β values 1.0 and 3.0 (the latter curves are not shown in the plot).

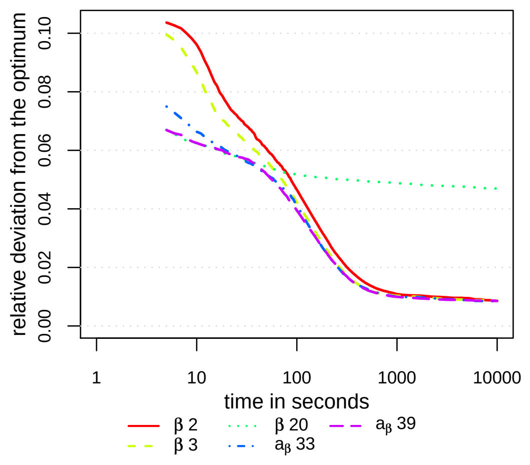
One more conclusion can be derived from figure 6.1(c). While $a_\beta\,40$ decreases β's values after 100 iterations, $a_\beta\,37$ and $a_\beta\,43$ do the same after 50, respectively 250 iterations. While the three curves still all approach the performance curve of a fixed $\beta = 1.0$, the different periods for which they stayed at $\beta = 20.0$ show some effect as well. The longer β stays at value 20, the longer the schedule's curve tracks the curve of fixed $\beta\,20$ (shown only in figure 6.1(a)). At the same time, the magic marker seems to have moved forward to some extent, possibly at the cost of final solution quality.

For future work, it would be interesting to investigate whether the magic marker can be pushed a lot more towards the beginning of the run. In fact, it is not really clear what the algorithm is doing during its initial phase without major improvements. One may argue that it takes some time, before the pheromone levels actually converge. It could also be possible that the pheromone distribution needs to adapt after a significant change in the value of β, before the algorithm is able to converge again. However, the latter is somewhat contrary to a previously stated observation. It does not really explain why staying with a high β for a longer period, and thereby effectively shortening the number of iterations the algorithm can adapt to the lower β, causes the magic marker to move forward. Further investigation is needed to explain this scenario.

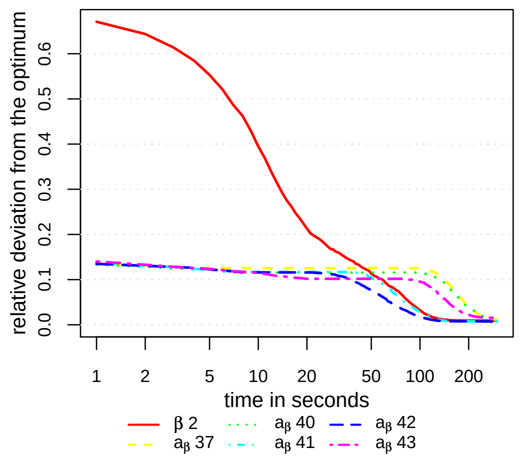
For the case with local search, the first of the above conclusions on the magic marker does not hold, i.e., the curves do not follow the trend of the respective fixed β curve once they cross it. Instead, schedules changing β in the same iteration seem to all follow the same trend (not shown here). Figure 6.2(d) illustrates the different algorithm behavior that can be observed based on the point in time, when the schedule steps down to the



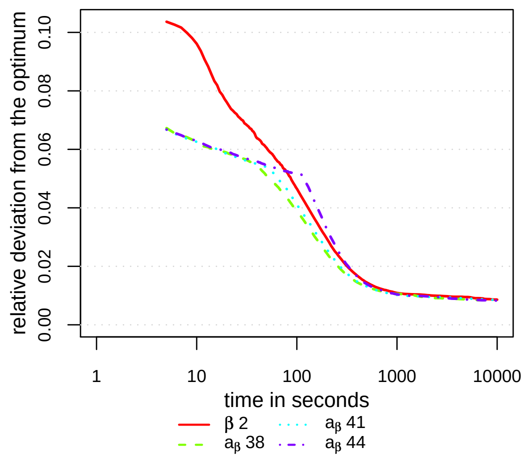
(a) instance 400-1: varying β, no local search



(b) instance 6000-1: varying β, 2-opt



(c) instance 400-1: varying β, no local search, focus schedule sensitivity



(d) instance 6000-1: varying β, 2-opt, focus schedule sensitivity

Figure 6.2: Performance plots using deterministic schedules to set β in MMAS

lower β. It is possible to determine the approximate moment when the switch occurs by using table 5.3. The latter provides an approximate mapping of runtime on the number of performed iterations.

Even though these results suggest, that decreasing β rather early shifts the magic marker to the left, further experiments switching beta after only a few iterations could not verify this assumption. The impression that this may not be possible is also supported by figure 5.2(b), illustrating the levels of solution quality that can be reached by varying β.

Concluding this section, it can be stated that schedules to adapt β during the run have been proven feasible and effective. However, it needs to be noted that their behavior is not yet fully understood. This may or may not lead to the discovery of additional optimization potential if examined further in the future.

6.3 Adapting the Evaporation Rate

The third parameter for which deterministic schedules were developed is ρ, i.e., the evaporation rate used in MMAS. When changing the parameter ρ during a run, other measures need to be adjusted as well. This is necessary, as τ_{max} is derived based on the value of ρ, and τ_{max} is used further to determine the value of τ_{min} (see equations 2.14, 2.22, and 2.23).

Figure 6.3 shows the performance of different schedules that were examined. Again, the upper two subplots illustrate the behavior of well-performing schedules, while the lower two present further schedule variants to illustrate relevant trade-offs.

ta_ρ 1: one-step scheme, starting at $\rho = 1.0$ and setting $\rho = 0.01$ after 2 iterations
a_ρ 14: random scheme, setting ρ based on a uniform distribution between 0.0 and 0.5
a_ρ 15: random scheme, setting ρ based on a uniform distribution between 0.0 and 0.1
a_ρ 16: random scheme, setting ρ based on a uniform distribution between 0.0 and 0.05
a_ρ 27: one-step scheme, starting at $\rho = 0.9$ and setting $\rho = 0.2$ after 100 iterations
a_ρ 28: one-step scheme, starting at $\rho = 0.6$ and setting $\rho = 0.02$ after 250 iterations
a_ρ 29: one-step scheme, starting at $\rho = 0.9$ and setting $\rho = 0.2$ after 250 iterations
a_ρ 31: linear schedule, starting at $\rho = 0.9$ and subtracting 0.01 every 5 iterations until $\rho = 0.2$
a_ρ 32: linear schedule, starting at $\rho = 0.6$ and subtracting 0.01 every 10 iterations until $\rho = 0.02$

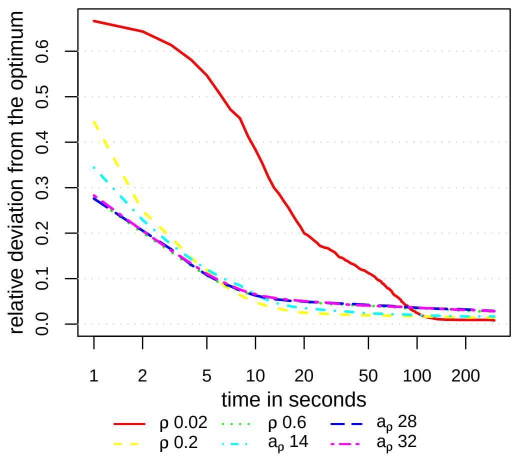
Considering runs without local search, one particular aspect distinguishes schedules for ρ from those for m and β. For the latter parameters, most well-performing schedules were typically able to improve the algorithm's performance in early stages of the run without any loss of solution quality towards the end. However, this turns out to be rather hard to achieve for ρ.

For cases without local search, the schedules a_ρ 28, respectively a_ρ 32 both start with a value for ρ that performs well in early stages of the run. Subsequently, they decrease the parameter to its standard value 0.02. While similar schedules for m and β perform well throughout a run, figure 6.3(a) shows that the two methods are not able to align with the reference configuration's curve when reaching it. Effectively, both curves keep tracking the curve corresponding to using their initial level of $\rho = 0.6$ throughout the whole run.

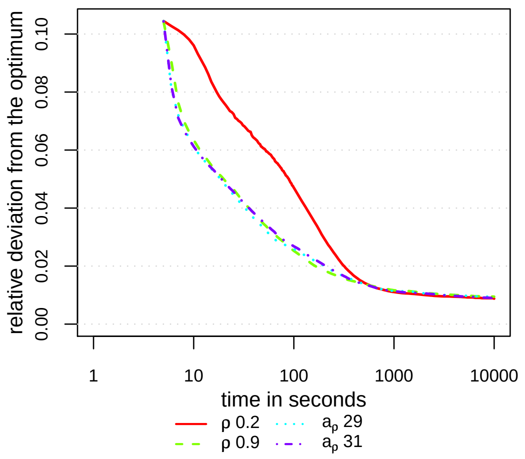
Surprisingly, setting ρ in a random manner is able to perform equally well in early stages while coming rather close to the reference configuration's curve with respect to final convergence. It should also be noted that choosing a fixed value of $\rho = 0.2$ performs quite well compared to the proposed schedules, which may question the use of schedules for ρ in a more general manner.

For runs with local search, final solution quality seems to be a less important issue. Quite reliably, the curves track the performance of the respective run with fixed ρ. Towards the end of the run, one can argue that the schedule a_ρ 31 performs somewhat better than the fixed setting of $\rho = 0.9$, which it otherwise tracks. It is this small difference that may justify the use of schedules for the parameter ρ when applying local search. Without it, using the respective fixed parameter setting would perform equally well.

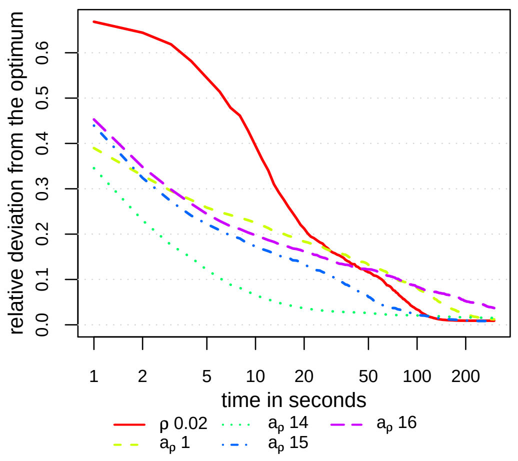
Figure 6.3(c) shows that without local search, there are actually variants of random and one-step schemes that are able to achieve the solution quality of the reference configuration (i.e., a_ρ 15 and a_ρ 1). While this advantage comes at the price of being somewhat inferior



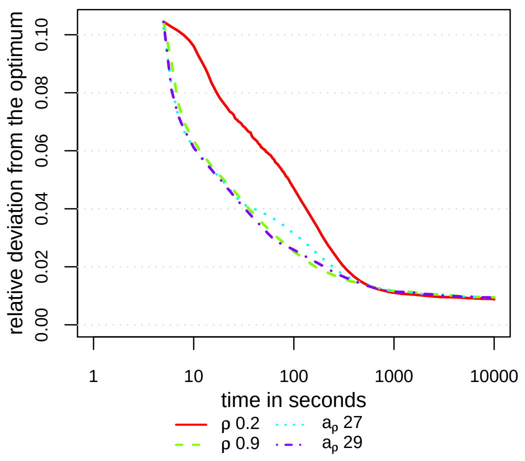
(a) instance 400-1: varying ρ, no local search



(b) instance 6000-1: varying ρ, 2-opt



(c) instance 400-1: varying ρ, no local search, focus schedule sensitivity



(d) instance 6000-1: varying ρ, 2-opt, focus schedule sensitivity

Figure 6.3: Performance plots using deterministic schedules to set ρ in MMAS

to the results shown in figure 6.3(a), at least the random scheme a_ρ 15 is still better than the reference configuration.

Figure 6.3(d) illustrates for configurations with local search that decreasing the evaporation rate too early causes the schedule's performance curve to deviate from its optimal path. Whenever the switch to the lower evaporation level occurs, the convergence process is slowed down and the curve somewhat flattens out for a while. While this is clearly visible for a_ρ 27, a_ρ 29 shows only a slight setback right before it crosses the curve using a fixed ρ of 0.9.

In summary, it can be stated that using deterministic schedules for the evaporation rate is generally applicable. However, for cases without local search, it is hard to find schemes that perform equally well throughout all stages of the run.

It should also be kept in mind that there are fixed settings for ρ that perform similarly well as the best among the schedules that were examined. Even though testing new schedule compositions is for sure an option, one may also want to consider choosing a good static value for ρ while focusing on changing other parameters during the run.

6.4 Adapting the Tour Construction's Elitism

The fourth and last parameter for which deterministic schedules were developed is q_0, i.e., the parameter determining the algorithm's degree of elitism when using the pseudorandom proportional rule during tour construction. The results obtained using the below set of methods are presented in figure 6.4:

a_{q_0} 1: one-step scheme, starting at $q_0 = 0.99$ and setting $q_0 = 0.0$ after 2 iterations

a_{q_0} 4: one-step scheme, starting at $q_0 = 0.99$ and setting $q_0 = 0.0$ after 50 iterations

a_{q_0} 8: one-step scheme, starting at $q_0 = 0.99$ and setting $q_0 = 0.0$ when about 100% of the initial pheromone level τ_0 have been evaporated

a_{q_0} 14: linear schedule, starting at $q_0 = 0.99$ and subtracting 0.001 every 2 iterations until $q_0 = 0$

a_{q_0} 17: random scheme, setting q_0 uniformly random between 0.0 and 1.0

a_{q_0} 19: random scheme, setting q_0 using a uniform binary distribution, i.e., $q_0 \in \{0, 1\}$

While most of the above schedules follow a straightforward methodology, a_{q_0} 8 may require a somewhat more detailed explanation. The number of iterations $iter_x$ after which $x \cdot 100\%$ of the initial pheromone level is evaporated, is computed as follows:

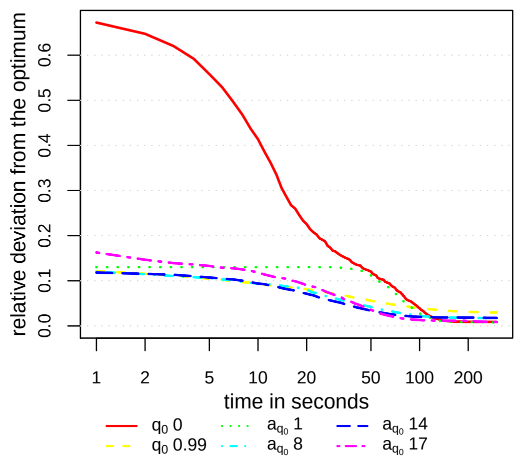
$$\tau_{min} + x \cdot (\tau_{max} - \tau_{min}) = \tau_{max} \cdot (1 - \rho)^{iter_x} \tag{6.3}$$

$$iter_x = \frac{log\left(x + (1 - x) \cdot \frac{\tau_{min}}{\tau_{max}}\right)}{log(1 - \rho)} \tag{6.4}$$

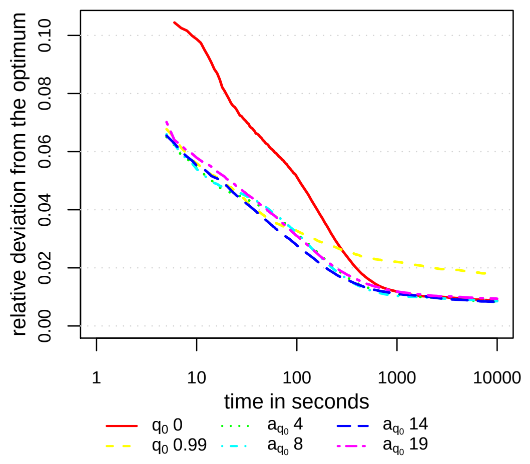
Equation 6.3 describes a situation in which $x \cdot 100\%$ of the possible evaporation has taken place, assuming an initialization of the pheromone levels at τ_{max} and an evaporation of $\rho \cdot 100\%$ each iteration. Pheromone deposits are not taken into account. Solving this equation towards $iter_x$ across several intermediate steps allows to derive the expression shown in formula 6.4. When the iteration counter reaches $iter_x$, the schedule a_{q_0} 8 would decrease q_0 to 0.0.

Figure 6.4(a) shows the best performing schedules for runs without local search. The first thing that becomes apparent is that compared to m and ρ, the q_0 schedules start at a relatively good solution quality. While there are schemes that are able to improve these good solutions even further right from the beginning of the run, many of them are charged a price in terms of final solution quality.

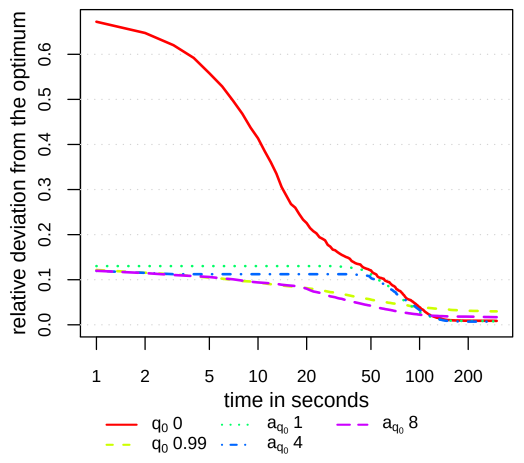
Two approaches seem to exist that allow starting at a significantly better solution without performing worse towards the end. The first one is to set q_0 to a relatively high value at the beginning and to decrease it rather quickly (see, e.g., a_{q_0} 1). The high initial q_0 implicitly enforces the construction of nearest neighbor tours, which allows the algorithm to start at a relatively good solution quality. Decreasing q_0 to a small level rather quickly seems to be necessary to not impact the final solution quality in a negative way. Generally, these schemes appear to follow a relatively flat line until they reach the curve of the respective low value setting, which they consecutively track. To some degree, this resembles the "magic marker situation" for the parameter β. Again, it may be interesting to investigate, whether the respective point can be moved further towards the beginning of the run.



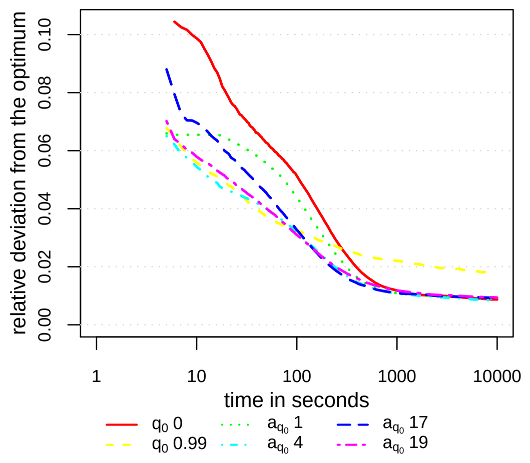
(a) instance 400-1: varying q_0, no local search



(b) instance 6000-1: varying q_0, 2-opt



(c) instance 400-1: varying q_0, no local search, focus schedule sensitivity



(d) instance 6000-1: varying q_0, 2-opt, focus schedule sensitivity

Figure 6.4: Performance plots using deterministic schedules to set q_0 in MMAS

The second promising approach may be a somewhat surprising one. The schedule a_{q_0} 17 sets a new random q_0 every iteration and seems to provide an interesting trade-off between performance in early stages and nearly no loss of solution quality towards the end.

With local search, the situation seems to be a somewhat different one. Linear and one-step schemes which decrease q_0 rather slowly respectively rather late all seem to be able to follow the trend of the best performing fixed value curves. Surprisingly, some of them even appear to improve over the reference configuration in later stages of the run. Random schemes again seem to achieve reasonable performance gains in early stages, but show some weakness towards the end.

Figures 6.4(c) and 6.4(d) show some variations of the previously examined schemes for runs without, respectively with local search. The former figure essentially demonstrates the effect of changing the length of the period after which q_0 is decreased. The longer the parameter stays on its initial value 0.99, the longer the method tracks the curve using this value throughout the run. If q_0 is set to 0.0 relatively early, the curve flattens out and does not converge significantly before meeting the reference configuration.

However, if q_0 stays at 0.99 rather long, as in a_{q_0} 8, a quite interesting observation can be made. From the algorithm's behavior with respect to the schedules a_{q_0} 1 and a_{q_0} 4 one would assume that its convergence stagnates once q_0 is decreased and that it does not recover until the curve for $q_0 = 0.0$ is met. But this is not the case. Once the parameter is set to zero (i.e., at about 20 seconds), it actually starts to improve compared to the curve it has been tracking before. Even though this advantage is somewhat offset by the method's poor performance towards the end of the run, it nevertheless demonstrates an interesting property of the parameter q_0. It appears that based on the prior history of algorithm settings, which effectively is depicted in the deposited pheromone, the search process reacts different to (the same) changes in q_0. This gives rise to the idea that there may be something like an optimal level of q_0 which is based on some property of the present pheromone levels. A similar idea will be investigated further in the context of the adaptive method proposed in chapter 7.

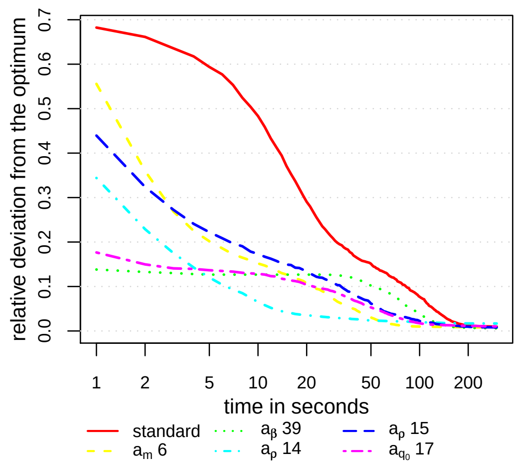
For the case with local search, figure 6.4(d) illustrates the effect of switching q_0 to a small value at two different points in time. It shows that using the high q_0 only for a short period is not sufficient to track the best possible curve. While switching q_0 to zero after only 2 iterations (i.e., a_{q_0} 1) causes a significant decrease in performance, doing the same after 50 iterations (i.e., a_{q_0} 4 at about 30 seconds) appears to barely have any effect at all.

The figure also shows the performance of two different random schedules. A binary random distribution performing better than a continuous one early on, while being slightly worse in later stages, leaves room for interpretation. One may argue that to achieve a good initial solution quality in terms of a nearest neighbor tour, extreme values of q_0 are required and thus, the binary distribution should perform better. In later stages, a too elitist tour construction may be somewhat harmful as it effectively hinders the exploration of the search space.

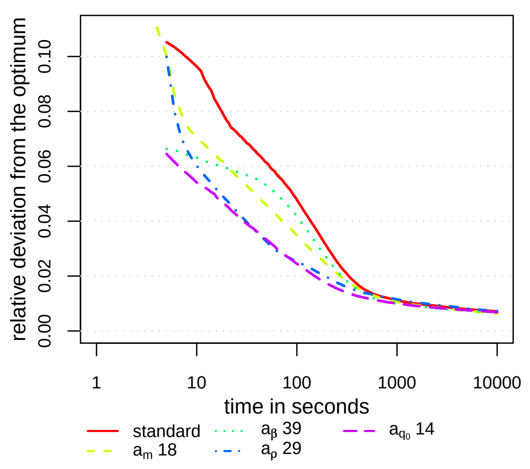
The experimental results presented in this section have shown that it is very well possible to create deterministic schedules for q_0 which improve algorithm performance. A good initial solution can reliably be created by choosing a high parameter value for some time. Without local search, a trade-off between further improvements in early, respectively late stages of the run may have to be taken into account. Random schemes show a surprisingly good performance for the examined instances. Even though not presented here for the sake of simplicity, sinus schemes for the parameter q_0 in some cases performed similarly well as random approaches.

6.5 Comparing Schedules for Different Parameters

This section aims at deriving first conclusions on which parameter, when being adapted using deterministic schedules, is able to improve the solution process the most. Of course, this does not account for potential benefits from changing several parameters at once, but it may give indications on which parameters to focus with respect to future work. Figure 6.5 shows the performance curve of the best schedules without, respectively with local search.



(a) instance 400-1: best schedules, no local search



(b) instance 6000-1: best schedules, 2-opt

Figure 6.5: Comparison of best-performing schedules for MMAS

Generally, the shown methods were selected with a special focus on final solution quality. The goal was to achieve a superior performance in early stages of the run without deteriorating results at the end.

For the case without local search (see figure 6.5(a)), all shown methods but $a_\rho\,14$ match this condition. The latter has been added as it is superior to all other methods during an intermediate stage of the run. Effectively, it shows what may be possible for this stage if one succeeds to further improve the remaining schedules.

What is especially noteworthy about the results for instance 400-1 is that several of the shown curves seem to be able to improve over the reference configuration during all stages of the run. While being clearly superior in early stages, some also appear to be slightly better towards the end. As all approaches show advantages in different stages of the run, it is hard to identify a single parameter that shows the largest potential for improving the algorithm's performance. The optimal choice for a specific problem setting without local search would essentially depend on the required trade-off with respect to the algorithm's performance across different runtimes.

Figure 6.5(b) shows the best-performing schedules for runs with local search. The schedules for q_0 and ρ are clearly superior to the others in early stages of the run. However, the latter pays a price in terms of final solution quality. All other methods perform slightly better than the reference configuration towards the end, the number of ants apparently being the most decisive factor in that regard. For now, the use of deterministic schedules for the parameter q_0 seems to offer the best trade-off between solution quality in early, respectively late stages of the run. This is effectively one of the reasons why changing q_0 on configurations with local search will be further examined in the following chapter.

One aspect should be noted about the results presented in figure 6.5. For three out of four parameters, random schemes were among the best-performing schedules. Either without or with local search, the parameters m, ρ, and q_0 are represented by methods using uniform random distributions to determine a new parameter value every iteration. This seems to be an indicator for the algorithm profiting from a certain degree of randomness being inserted into the search process. It may be a promising idea to examine combining a rather stable (e.g., linear) schedule for one parameter with a random scheme for another, thereby trying to profit from both approaches' strengths.

When considering which method to combine with other parameter control approaches, it is import to not only look at the respective performance curves. One should also consider the implicit effect of the different methods with respect to the development of the pheromone levels. E.g., there are several ways to enforce a good initial solution quality, which have quite different effects on the pheromone distribution. Choosing a high q_0 for a few iterations creates good quality solutions in early stages by enforcing the construction of nearest neighbor tours. Doing this leaves the pheromone levels largely untouched. Choosing a high evaporation rate in early stages also leads to the algorithm constructing good solutions rather soon. But in this case, the effect is due to the generally low and rather different pheromone levels, which the approach creates. These lead to an increased importance of the heuristic value for rarely selected edges, while at the same time making the algorithm more elitist with respect to recently deposited pheromone.

There is one more thing to be kept in mind when evaluating the performance of the different schedules for applications with local search. Chapter 5.2 previously showed that the results for schedules changing the parameter ρ are only slightly superior to those of choosing the best fixed ρ. Would it not be for the loss of solution quality towards the end, choosing a different fixed evaporation rate may be able to achieve a performance comparable to good deterministic schedules. However, even given this somewhat questionable situation, the concepts examined in this chapter provide insight into how the different parameters impact the algorithm's behavior. Methods were suggested that generally allow to track the best fixed settings' performance. Even if using a different evaporation rate in the reference configuration turned out to significantly improve its performance, applying similar schedules in this new region of the parameter space should again provide better results.

7 An Adaptive MAX-MIN Ant System Variant: MMASdde

In contrast to the deterministic parameter control methods examined in the previous chapter, the approach introduced in the following is of adaptive nature. Instead of choosing the respective parameter's value as a function of the iteration counter, it is determined based on specific properties of the search process. The new method is called *MAX-MIN Ant System with distance dependent elitism* (MMASdde) and derives its name from setting the degree of elitism used in tour construction (i.e., the parameter q_0) based on the average distance between the tours generated by the ants. A detailed explanation of this approach is given in section 7.1. The second section presents the results obtained from experimental analysis. The last section illustrates some considerations on how to derive an optimal average distance between tours (i.e., a measure to be introduced in the subsequent section) based on the size of the considered problem instance.

7.1 Method

The method described in the following paragraphs is based on the idea of an optimal degree of heterogeneity among the solutions generated in a particular iteration. It is assumed that if the tours created by the ants are too similar, the algorithm is not exploring enough; if they are too dissimilar, the algorithm does not exploit its generated knowledge, represented in the form of the pheromone levels, enough. To put this idea into practice, several subproblems need to be solved. First, one needs to find a measure of the heterogeneity of the solutions created in a specific iteration. Second, one needs to check whether specific values of the measure can be related to a particularly good, respectively bad performance of the algorithm. In other words, one needs to find out, whether there is something like an optimal level of heterogeneity, which one could try to maintain in order to sustain a high level of algorithm performance throughout the run. Finally, one needs to specify a rule set relating the value of the considered parameter to the used measure. This is necessary to be able to dynamically adapt the parameter so that the measure's value stays close to its optimal value. The subsequent paragraphs will illustrate how these steps were put into practice for the method examined in this chapter.

As measure of solution heterogeneity, the average pairwise distance of the tours constructed by the ants has been chosen. The distance between a tour T^a and a tour T^b is defined as the number of edges that are contained in T^a, but not in T^b. This is effectively the same as the number of cities n minus the number of common edges. While similar solutions share many edges and thus get assigned a small distance, rather different solutions accordingly have a high distance. Looking at the average of this value across all possible

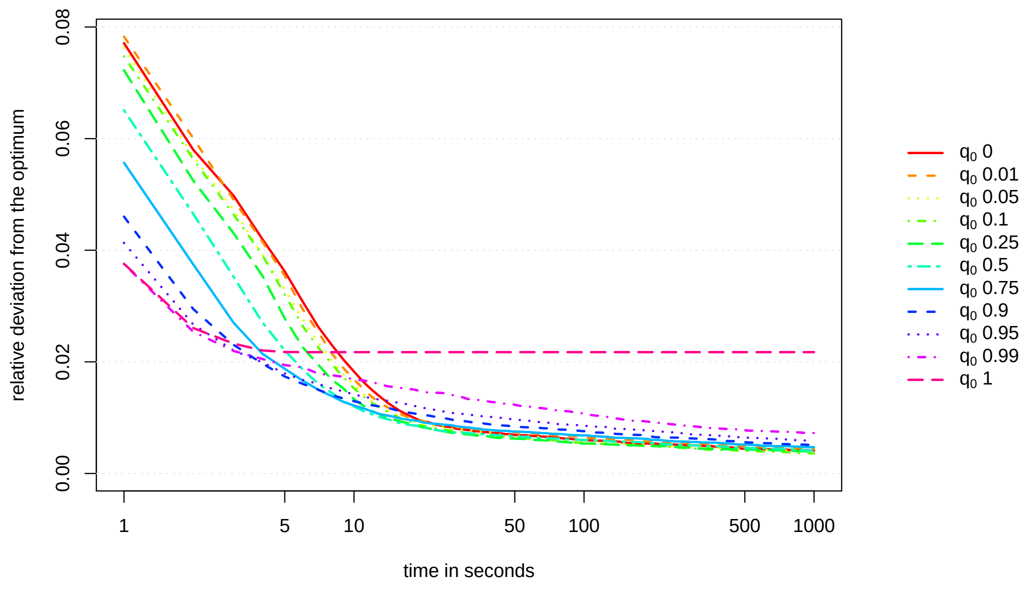


Figure 7.1: Performance plot for different q_0 values in MMAS, instance 1500-1, 2-opt

pairwise combinations of ants' tours indicates the algorithm's performance with respect to the trade-off exploration vs. exploitation.

To determine whether specific average distances perform superior to others, runs with different fixed parameter settings were compared to one another (i.e., similar to section 5.2). The goal was to identify a particular distance value that would be encountered for each setting of the examined parameter, each time marking a phase in which the respective setting performs better than the others.

Figures 7.1 and 7.2 illustrate this for different fixed values of the parameter q_0 with local search applied. While figure 7.1 shows the performance of the different configurations over time, the latter displays the corresponding development of the average distance. As the latter information has been recorded by iteration, one may want to use table 5.3 to get an impression of the development of distance over time. Two more things should be noted with respect to figure 7.2. Firstly, it displays only distance values larger than 1.0, which allows for a reasonable scaling on the logarithmic vertical axis. Secondly, all shown curves display some type of running average of the original data, i.e., the later in the run, the more iterations have been summarized into one data point. The non-flattened curves would be harder to analyze as they show a stronger fluctuation around the depicted lines. The same applies for consecutively shown plots of the average distance between solutions.

Using the data underlying the two figures, it has been possible to identify a particular value of q_0, which seemed to characterize the best performing curve across the whole range of runtimes. More information on how to determine, respectively predict this value is given in section 7.3. For now, one may consider 18 and 40 as the relevant values representing the results of a first manual analysis for problem instances 1500-1 and 6000-1. These are the target distances that were used to produce the results presented in the next section, i.e., the method will try to adapt q_0 in a manner that aims at keeping the solutions' average distance at this level.

The respective changes in q_0 are typically governed by a set of rules, increasing or decreasing the parameter based on the observed average distance. In the present case (i.e.,

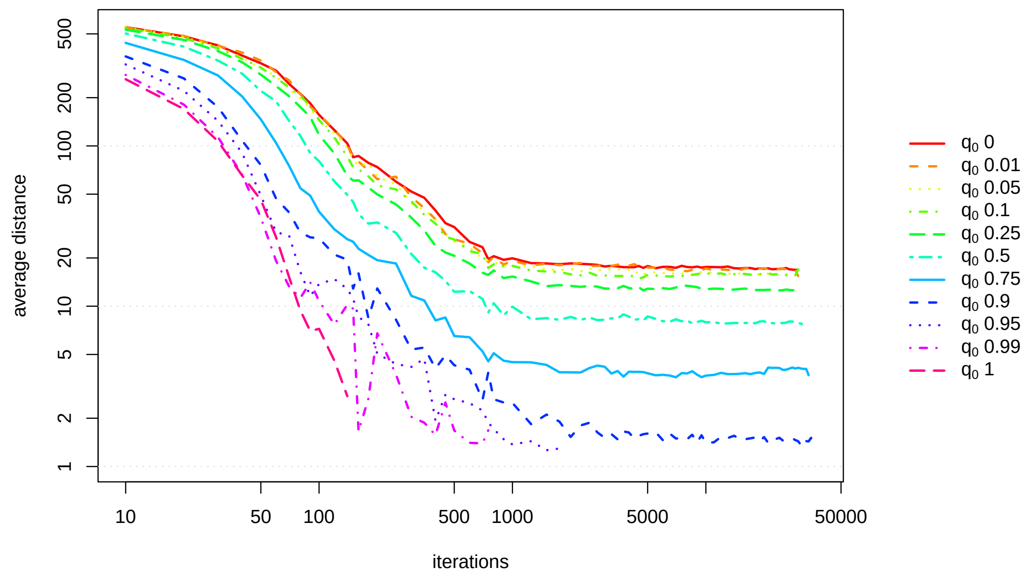


Figure 7.2: Distance plot for different q_0 values in MMAS, instance 1500-1, 2-opt

changing q_0; with local search), these rules are of rather simple nature. Considering the meaning of q_0 for the algorithm, it is intuitive that increasing it would decrease the average distance and vice versa. Increasing the degree of elitism with respect to tour construction favors a rather small group of well performing edges and thus leads to the algorithm producing relatively similar solutions. Using low elitism on the contrary increases the chances of less visited edges to become part of a tour and thus increases solution diversity. This intuition is confirmed by figure 7.2, which suggests that the higher one chooses q_0, the lower is the average distance.

It needs to be noted that the method as presented here can be applied only for runs with local search. Other configurations focusing on changing β, respectively ρ, both, for cases with and without local search, have also been subject to a first investigation. However, there either seemed to be no clear optimal distance value for these configurations or the rule set required to track it appeared more complex than the one above. A similar setup for g_{tau} has been discarded as changing the parameter showed too little impact on the algorithm's behavior (see subsection 5.2.5).

One question that stayed unanswered so far is how to adapt q_0. Generally, linear or exponential stepping would be applicable (i.e., changing q_0 by a fixed amount or multiplying it with a fixed factor). However, these schemes leave only few options to change the variable's behavior in more complex ways. Therefore, the method as examined here uses fixed sets of possible values for q_0 (i.e., q_0 stepping schemes). Each time, the variable is changed, the next larger, respectively smaller value from the respective set is chosen. Table 7.1 gives an overview of the sets that were used in this work.

For each set of values, there are two adaptation schemes a_{q_0} using the set. One of them starts the parameter adaptation at the highest value of the set (e.g., a_{q_0} 900), the other one at the lowest value (e.g., a_{q_0} 910). While the former configuration is the more intuitive approach, the latter one aims at demonstrating the ability of the adaptive process to cope with a suboptimal initialization.

a_{q_0}	set of used q_0 steps	initial q_0
900 (910)	$\{0.0, 0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99, 1.0\}$	1.0 (0.0)
901 (911)	$\{0.0, 0.01, 0.1, 0.5, 0.9, 0.99, 1.0\}$	1.0 (0.0)
902 (912)	$\{0.0, 0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1.0\}$	1.0 (0.0)
903 (913)	$\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$	1.0 (0.0)
904 (914)	$\{0.0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.07, 0.1, 0.25, 0.5, 0.75, 1.0\}$	1.0 (0.0)

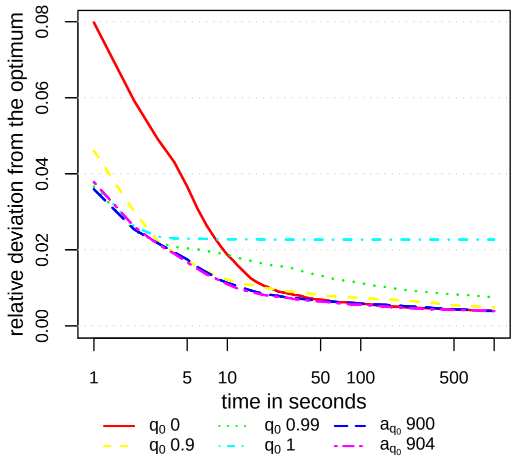
Table 7.1: Stepping schemes for q_0 as used by MMASdde

The different sets of potential q_0 values were chosen to examine the impact of focusing on different regions of the parameter's permissible value range. Some of them choose more values close to one or both ends of the possible range, while others use a more evenly distributed set of values. By choosing a rather large set of values close to zero, the schemes a_{q_0} 904 and a_{q_0} 914 try to trap the adaptation in the lower ranges. I.e., once q_0 has reached a rather small level, the parameter value is significantly increased only if the average solution distance stays above the target value during several consecutive adaptation steps.

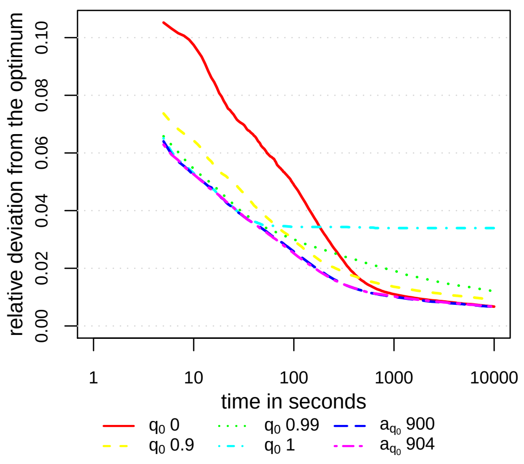
Two more aspects should be noted with respect to the experimental setup used in this chapter. Firstly, the subsequently presented results were obtained with MMAS pheromone re-initialization disabled (i.e., $iter^{stag}_{min} = \infty$). This design choice has been made upfront as the respective rapid changes in the pheromone levels were expected to potentially deteriorate the adaptive method's performance. With hindsight of the obtained results, the method turned out to be rather robust and should generally be able to handle such changes. Thus, future extensions of this work should no longer make this restriction and instead use a standard MMAS setup.

The second aspect to be noted is concerned with the frequency at which the adaptation of q_0 takes place. As the computation of the average solution distance is somewhat computational expensive (i.e., $O(n^2)$), the adaptation is performed only every 10 iterations in the standard configuration. The effect of adapting q_0 every iteration is also briefly reviewed in the next section.

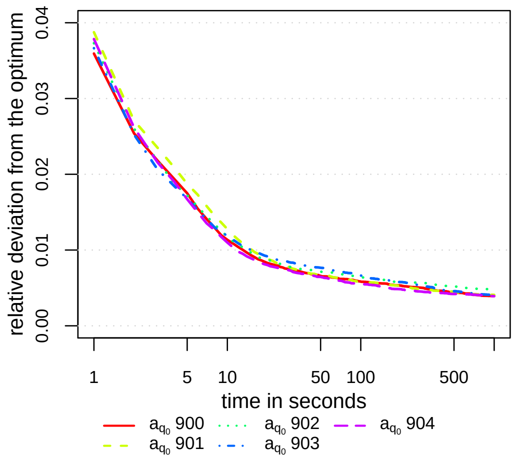
For a better understanding of how MMASdde relates to the concepts introduced in chapter 2, it may be interesting to once again review the pseudo-code introduced on page 5. An implementation of the adaptive method requires to extend the code towards the end of the algorithm's main loop (i.e., at any point after the application of local search). If the iteration counter is a multiple of the chosen update frequency, two additional steps have to be executed. First, the average distance between the tours constructed by the ants needs to be determined. Then, based on how the result relates to the chosen target distance, q_0 would either be decreased or increased. The new value of q_0 would be used in subsequent iterations of the main loop, until the next adaptation of the parameter takes place. The initialization phase would be responsible for assigning q_0's initial value as per table 7.1.



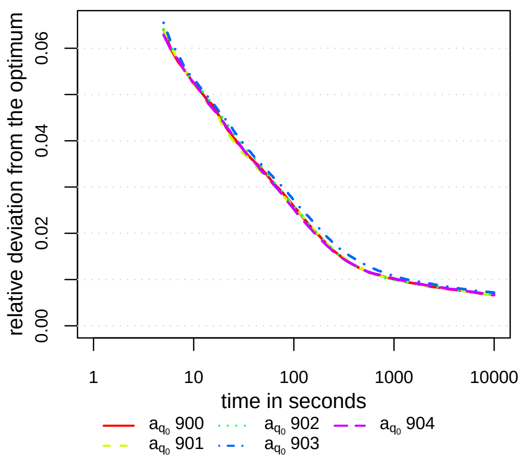
(a) instance 1500-1: varying q_0, 2-opt



(b) instance 6000-1: varying q_0, 2-opt



(c) instance 1500-1: varying q_0, 2-opt, focus set sensitivity



(d) instance 6000-1: varying q_0, 2-opt, focus set sensitivity

Figure 7.3: Performance plots using a distance-based adaptive method to set q_0 in MMAS

7.2 Results

The results of the analysis described in the previous section are depicted in figure 7.3. The upper two plots each compare two selected variants of the adaptive approach to well performing static configurations. The lower two compare all five previously proposed variants starting with a q_0 of 1.0 (see table 7.1). As the method is concerned only with configurations using local search, the instances 1500-1 and 6000-1 were chosen to give the reader an idea of the impact of instance size.

Generally, the well performing variants a_{q_0} 900 and a_{q_0} 904 are at all times able to track the best-performing curve corresponding to some static configuration. The results also show that independent from the chosen q_0 stepping scheme, the adaptive process is able to follow the same general trend. There are slight indications that concentrating more values around the lower and possibly also the upper end of the value range seems profitable. However, the differences in performance as observed among the five adaptive algorithm variants still lack a more conclusive explanation. Comparing the results of the two examined instances suggests that the chosen q_0 stepping scheme becomes rather irrelevant as the instance size increases.

Figure 7.4 analyzes the behavior of the proposed adaptive process in more detail using problem instance 1500-1. Equivalent plots for instance 6000-1 were generated and confirm the subsequently discussed behavior. While the left three plots in figure 7.4 show information about the standard configuration, which updates q_0 all ten iterations, the plots on the right show the behavior with updates in every iteration. All plots show curves for three different configurations, i.e., the reference configuration and two of the adaptive approaches presented in this chapter. While one of the adaptive methods (i.e., a_{q_0} 900) starts with a q_0 of 1.0, the other one (i.e., a_{q_0} 910) starts with q_0 equal to 0.0.

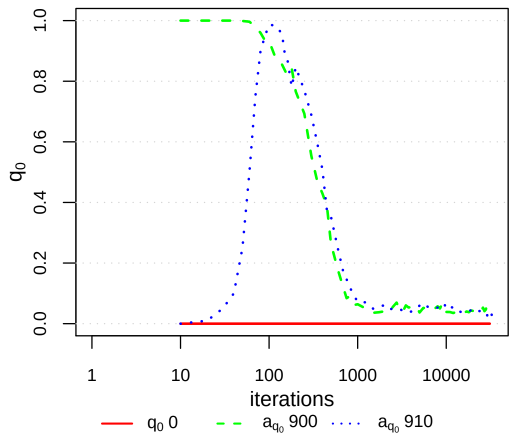
Figures 7.4(a) and 7.4(b) show the development of q_0 during the run. After an initial phase in which the development of the parameter is depicted in an exact manner, the data points in later stages show an average across several iterations in order to improve the readability of the plot. The curves clearly show how the method assigns a high q_0 in early stages and subsequently decreases it in the course of the run. Even if starting at a low q_0 value, as with a_{q_0} 910, the method is able to catch up rather fast. One may be able to argue that adapting q_0 only every ten iterations causes a small delay with respect to when the parameter is decreased again. This trend does not seem to persist in a noticeable manner when adapting every iteration.

The next two plots in the same figure show the development of the average solution distance for the same three runs. The curves show that the two adaptive approaches reliably converge to the target distance (i.e., 18 for instance 1500-1). By comparing the development of q_0 and the average solution distance over time, one can confirm that the method works as expected. As long as the average distances are rather large, a high q_0 is used. As the distance converges to its target value, q_0 is decreased to a rather low value.

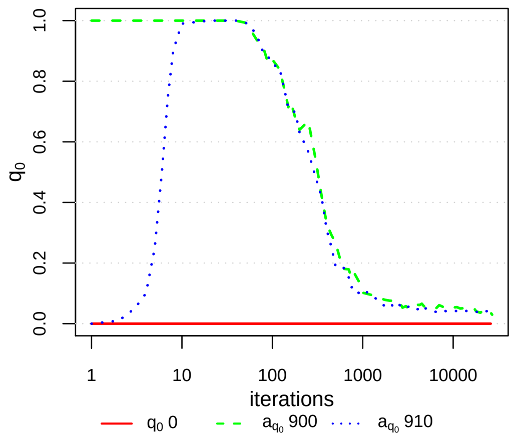
The performance for the two update frequencies is depicted in figures 7.4(e) and 7.4(f). Generally, both adaptive settings are able to improve over the reference configuration. However, the adaptive method starting at $q_0 = 0.0$ (i.e., a_{q_0} 910) needs some time until it is able to catch up with the configuration starting at the optimal q_0 (i.e., a_{q_0} 900). With respect to the last two plots, it becomes clear that the update frequency has a crucial effect on performance if a rather large number of adaptation steps are required before q_0 reaches its optimal level for the first time.

Along these lines, it may be worth considering one particular idea for a future extension of the used methodology. Alternative to increasing the frequency of parameter updates, one could try to relate the magnitude of the change to the magnitude of the discrepancy between the present solution distance and the target distance. I.e., if the observed average distance is significantly larger than the target distance, one could increase q_0 by several steps instead of just one. One possible way to do this could be to increase, respectively decrease q_0 by at maximum x steps, if the average distance misses its target value by a factor 2^x. Such an approach would not only solve the problem of a sub-optimal initialization, it could also help the algorithm to react more appropriately to changes in distance that occur during later stages of the run. However, one should be aware that in the latter case, overshooting the required q_0 may cause the algorithm to oscillate around the optimal distance value if not dealt with correctly.

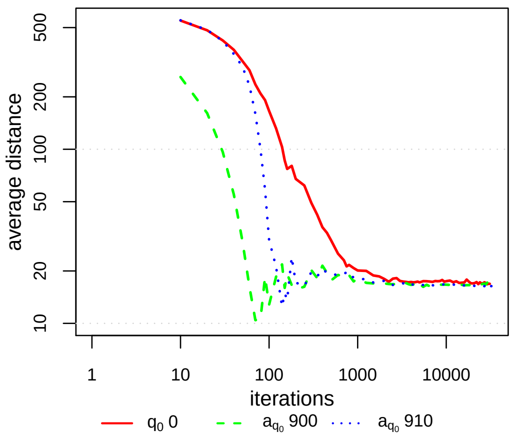
A last interesting comparison to be made is the one contrasting the performance of deterministic schedules with the performance of the new adaptive approach. Figure 7.5



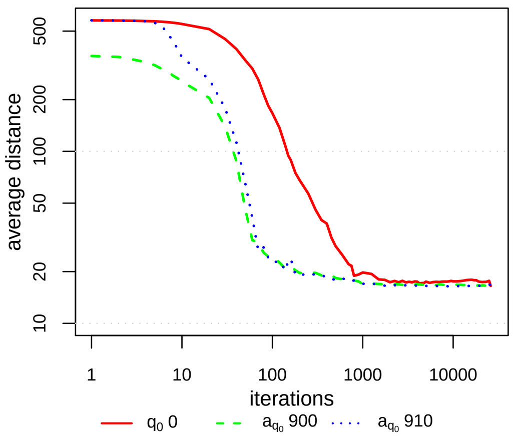
(a) development of q_0, update every 10 it.



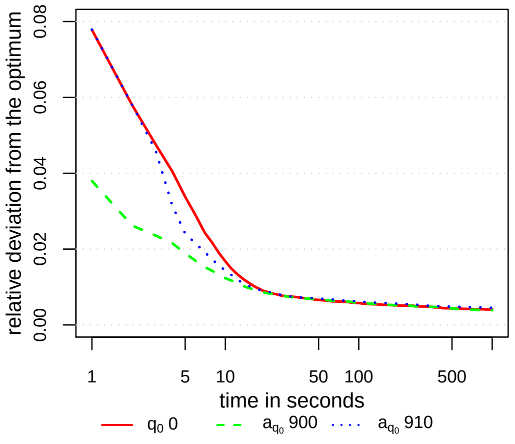
(b) development of q_0, update every it.



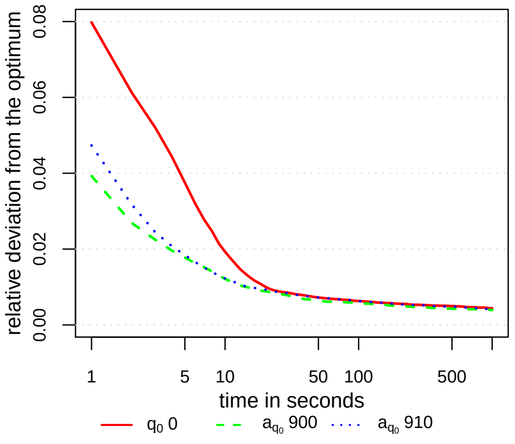
(c) distance plot, update every 10 it.



(d) distance plot, update every it.

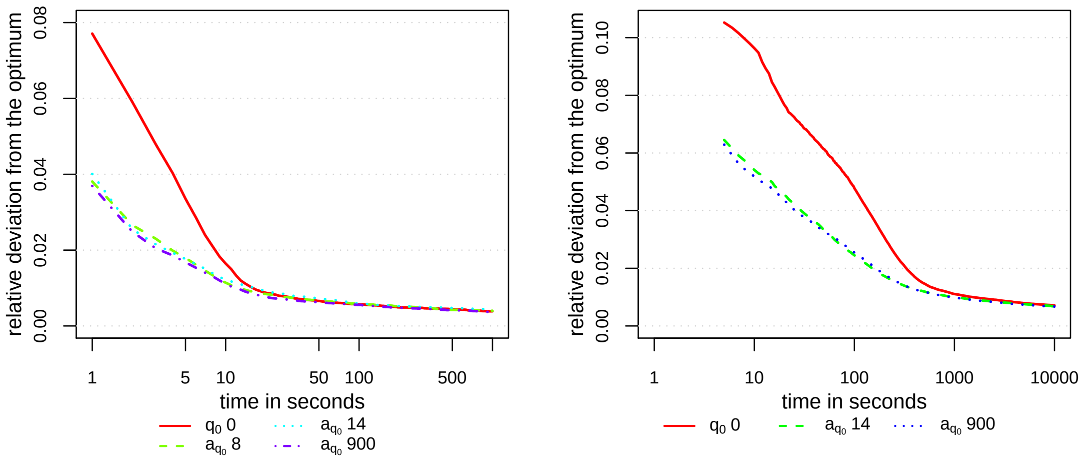


(e) performance plot, update every 10 it.



(f) performance plot, update every it.

Figure 7.4: Effect of q_0 starting value when using a distance-based adaptive method to set q_0 in MMAS, instance 1500-1, 2-opt



(a) instance 1500-1: varying q_0, 2-opt

(b) instance 6000-1: varying q_0, 2-opt

Figure 7.5: Comparison of best-performing schedule with adaptive q_0 for MMAS

shows the respective curves for the instances 1500-1 and 6000-1. It turns out that both approaches perform more or less equally well, possibly with slight advantages for the adaptive method.

Still, there are further advantages that come along with the use of an adaptive methodology. Assuming that an appropriate target distance is known or can easily be derived for a particular class of problem instances, the method is expected to be able to dynamically adapt to various states of the search process. It has previously been shown that it is, e.g., able to cope with different initialization conditions. The next section proposes a simple method to predict approximate target distances for the uniformly random problem instances used in this work. Considering the option to transfer the proposed methodology to optimization problems other than the TSP, using an adaptive approach may even become a necessity. Some problem settings may require q_0 to continuously undergo changes throughout the whole run. This may be due to a higher inherent dynamic of the search process for some static problems or it may apply to explicitly dynamic problems whose fitness landscape changes during a run.

7.3 The Target Distance: Further Considerations

This section proposes an approach to determine the optimal average solution distance (i.e., the target distance) in a more structured way. While the target distances used to generate the presented results were intuitively selected based on comparing different parameter settings' distance development, this section aims at providing a more reproducible approach.

The idea underlying this approach is rather simple. It is based on the assumption that towards the end of the run, the reference configuration (i.e., $q_0 = 0$) is typically the best performing configuration with a fixed q_0 value. Figure 7.2 has shown before that the reference configuration's average distance measure typically converges to a rather stable value as the algorithm run progresses. Examining the curves of the other configurations using a fixed q_0 value, it turns out that they all cross that particular value at some point.

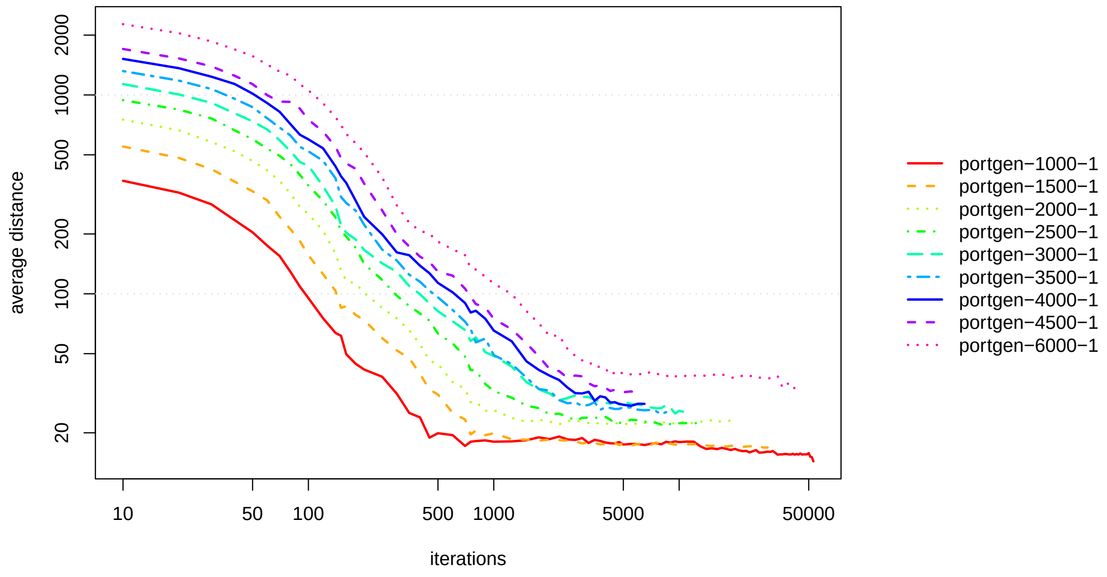


Figure 7.6: Distance plot for the MMAS reference configuration on different problem sizes

Typically, this crossing of lines happens with a time-wise proximity to phases in which the particular setting performs especially well. Thus, the average solution distance as realized by the reference configuration towards the end of the run is herein considered the optimal distance.

Figure 7.6 shows how the distance measure develops for different problem sizes when using the reference configuration. The curves' different lengths origin from different run-times, respectively the higher duration of a single iteration with larger problem sizes. Table 7.2 lists the average distances that have been determined for the different problem instances, as well as the iterations that are reflected in these averages. Based on these average distances, a linear regression on the size of the problem instances has been performed. The values predicted by the linear regression and their deviation from the actual values are also depicted in the table.

instance	average distance	considered iterations	distance predicted by linear regression	regression residual
1000-1	15.62	45,001-50,000	16.72	-1.10
1500-1	17.10	23,001-28,000	18.67	-1.57
2000-1	22.89	13,001-18,000	20.62	2.27
2500-1	22.14	8,001-12,000	22.57	-0.43
3000-1	26.34	7,501-10,000	24.52	1.82
3500-1	25.74	5,501-8,000	26.47	-0.73
4000-1	27.67	4,751-6,000	28.42	-0.75
4500-1	32.35	4,501-5,500	30.37	1.98
6000-1	34.77	35,001-40,000	36.22	-1.45

Table 7.2: Average solution distance at the end of the run, using the MMAS reference configuration on different problem sizes: experimental results and linear regression

According to the linear regression, the average distance of the reference configuration, respectively the target distance d_{avg}^{target} can be derived as:

$$d_{avg}^{target} = 12.8243575 + 0.00390003 \cdot n \tag{7.1}$$

It may be a legitimate simplification to state that a first estimate of the average distance goal can be derived by taking a value of 13 and adding another 4 for every $1,000$ cities of problem size. To derive a more accurate estimate, one may have to examine additional problem instances, respectively one may want to use longer runtimes to ensure the average distances are converged in all cases.

However, first results indicate that MMASdde is not extremely sensitive with respect to the used target distance. It rather seems to be the order of magnitude that matters in the first place.

It needs to be noted that the optimal values suggested by the above methodology are expected to be dependent on the used reference configuration, especially on the number of ants and the used local search method. First experimental results also seem to indicate that a clustering of the used problem instances may limit the method's applicability in its present form. Before blindly using the method in future analysis, one may want to confirm its results in the respective context. The general idea of deriving the optimal distance from the curve evolving the highest final solution quality may nevertheless be transferable to rather different configurations.

8 Conclusion

8.1 Summary

This work dealt with improving ACO algorithms' performance by changing their parameter settings online, i.e., while the algorithm is being executed. After outlining four of the most prominent ACO algorithms for the TSP, prior work in related fields has been reviewed. Parameter setting methods introduced for Evolutionary Algorithms were examined with respect to their applicability in an ACO context and work done in the latter field has been discussed.

A first experimental analysis presented in this work focused on the performance of MAX-MIN Ant System with different static parameter settings. This study started by a previously defined reference configuration and changed only one parameter at a time. Typically, the best performing parameter setting is dependent on the point in time at which the solution quality is measured. While some settings quickly obtain a relatively good solution but stagnate only little later, others perform worse in the beginning but improve over the former in later stages. These observations effectively provided some intuition on how to change the respective parameters over time in order to have the algorithm achieve a good solution quality during all stages of the run.

Based on this intuition, chapter 6 proposed deterministic schedules to vary the parameters m, β, ρ, and q_0 during a run. Computational experiments proved the schedules' applicability in the given context. Nearly all presented methods are able to improve the reference configuration's performance in early stages of the run. Especially the schedules proposed for β and q_0 yield significant improvements in the first seconds. Some approaches achieve the good performance in early stages at the cost of being inferior to the reference configuration later on. However, other schedules are even able to improve the performance towards the end of the run. This is especially noteworthy, as this is where the reference configuration is typically among the best-performing setups.

Another aspect that should not go unnoticed is that some rather unintuitive schemes showed an impressive performance. In several cases, assigning a new randomly generated m, ρ, or q_0 every iteration seemed to introduce a useful diversity into the search process.

For varying q_0 in configurations with local search, one more parameter control method was introduced in chapter 7: MAX-MIN Ant System with distance dependent elitism (MMASdde). It varies the tour construction's degree of elitism (i.e., q_0) in an adaptive manner, i.e., its value is determined taking information about the state of the search into account. The new method is based on the idea that the algorithm performs at its best, if the tours constructed by the ants show a specific degree of heterogeneity. Using the average solution distance as a measure for the latter, q_0 is adapted to keep the measure on its target value.

The results of a computational analysis showed that the adaptive method works as expected. It is able to reliably steer the search process towards meeting the target distance and to subsequently maintain the desired level of heterogeneity. Even when starting at rather unfavorable settings for q_0, the new approach quickly adapts accordingly. The performance of MMASdde is comparable to the one of good deterministic schedules for the same parameter.

For the experimental setup as used in the present work, section 7.3 proposed a method to determine appropriate target distances as a function of the problem size. Given this additional information, MMASdde is expected to be rather robust and may be worth further investigation in the future.

8.2 Options for Future Research

In the course of this work, a series of opportunities for future research have been pointed out. This section will focus on ideas directly related to the present work. Further possible approaches inspired from previous research were pointed out in the literature review in section 3.2. From the author's perspective, the potential applications for operator selection methods (see subsection 3.2.3) deserve particular attention.

With respect to examining fixed parameter settings, a main focus should be to identify further local optima in the parameter space. Tools like iterated F-Race or a smart full factorial test design may permit to find better starting points for a potential parameter adaptation and may help to understand trade-offs relating to more than one parameter. For the reference configuration as presented in subsection 4.2.1, choosing a somewhat higher β or using a small positive q_0 promises to improve performance. Decreasing the population size for cases without local search, respectively increasing it when local search is applied, also seems to yield advantages in early, respectively later stages of the run.

Concerning deterministic schedules, it may be a reasonable next step to investigate possible benefits from adapting several parameters at once. To successfully configure schedules for several parameters, it will probably be necessary to take interactions between the different parameters into account. Simply running two or more of the previously examined schedules at once may very well lead to different effects partially offsetting one another, but should nevertheless be investigated before turning to more complex studies.

Another interesting research direction could be to further examine the notion of useful randomness as suggested by the success of random parameter control schemes. One could, e.g., try to combine a rather stable (e.g., linear) approach for one parameter with a random method for another parameter to identify the scenarios in which this form of noise has a positive impact on the search process. It may also be possible to develop approaches using a variable degree of randomness for a particular parameter and to thereby control measures such as the average solution distance. One could, e.g., try to track a target distance by changing a uniform random distribution's value range in an adaptive manner.

The newly introduced MMASdde could also be an interesting subject for further analysis. In a first step, one could explore the algorithm's behavior when using different target distances. This would allow to derive further conclusions on the method's sensitivity with respect to this new parameter. It would also help assessing whether the method proposed in section 7.3 is able to reliably derive well-performing values for the target distance.

In a next step, one could try to apply similar concepts to other parameters, respectively to configurations without local search. An initial examination of the average solution distance for different levels of q_0 without the application of local search showed some potential for adaptation. However, the rule set required to steer the adaptive process seemed somewhat more complex than the one presented in the previous chapter. One may also have to limit the value range used for q_0 in this case.

From a more general perspective, it may be of interest to investigate how different problem characteristics impact the performance of the proposed methods, e.g., by running experiments on clustered TSP instances.

Another idea that opens a wide field of research is the application of the concepts proposed in the present work to contexts other than the TSP. E.g., results by Randall [59] indicated that the QAP may profit more from adapting parameters during the run than does the TSP. One may also want to consider choosing a problem class where ACO approaches are among the best-performing methods and where parameter control methods could thus help to achieve or improve state-of-the-art performance.

To close a section on future research opportunities, it may be appropriate to recap a related statement of one of the pioneers in the broader context of the field. In a recently published book chapter, Ken de Jong [16] states that from his perspective, the most promising applications of parameter control methods can be found in dynamic environments. It is intuitive to argue that it is more natural (and should be more rewarding) to apply such methods to problem settings that are by definition changing their fitness landscape over time and thus inherently require an adaptive behavior on the parameter side. Assuming an increasing real life importance of dynamic optimization problems in the next decades, the author believes that the future focus of the research community is likely to somewhat shift in that direction.

8.3 Acknowledgments

The author especially wants to acknowledge the help of his supervisor Thomas Stützle with respect to the present work. He supported the creation of this thesis in numerous discussions and by providing inspiring guidance towards scientific work.

Further thanks are directed towards Manuel López-Ibáñez, Marco A. Montez de Oca, Mauro Birattari, and many other members of the IRIDIA lab that supported this work with their suggestions and comments.

Finally, together with Thomas Stützle, Marco Dorigo deserves gratitude for enabling the author's research stay at the Université Libre de Bruxelles (ULB) and for supporting the present work by providing access to the lab's computational resources.

Overview of Used Variables and Sets

Used Variables

α	relative influence of the pheromone trail during tour construction
avg	average number of different choices available to an ant at each step while constructing a solution
$best$	alias for the ant, resp. tour used in the pheromone update
β	relative influence of the heuristic values during tour construction
bs	alias for the best-so-far ant, resp. tour
$cand$	number of cities on the candidate list
C^k	cost, resp. length of the tour built by ant k
C^{nn}	cost, resp. length of a nearest neighbor tour
d_{ij}	distance from city i to city j
d_{avg}^{target}	average distance target used by MMASdde
η_{ij}	heuristic value of the edge between city i and city j
f_λ^i	λ-branching factor of city $i \in N$
f_λ^{avg}	average λ-branching factor of all cities $i \in N$
$f_\lambda^{restart}$	threshold with respect to f_λ^{avg} that in combination with other factors triggers a pheromone re-initialization in MMAS
g_τ	factor determining the size of τ_{min} relative to τ_{max} for $s \in S_{ls}$
ib	alias for the iteration-best ant, resp. tour
$iter_{min}^{stag}$	minimum number of iterations without improvement before a restart may be triggered
λ	sensitivity of the λ-branching factor
m	number of ants
n	number of cities
${}_{pp}p_{ij}^k$	probability with which ant k, located at city i, continues its tour to city j according to the pseudorandom proportional rule
${}_{rp}p_{ij}^k$	probability with which ant k, located at city i, continues its tour to city j according to the random proportional rule

q_0	probability of simply choosing the *best* city according to the random proportional rule instead of applying the rule as described for AS
rb	alias for the restart-best ant, resp. tour
ρ	pheromone evaporation level for the global pheromone update
τ_0	initial pheromone level
τ_{ij}	pheromone on edge between cities i and j
τ_{ij}^p	value of τ_{ij} prior to applying a specific update
$\Delta\tau_{ij}^k$	amount of pheromone deposited by ant k on the edge between cities i and j
τ_{min}	minimal pheromone value permitted on any edge
τ_{max}	maximal pheromone value permitted on any edge
τ_{min}^i	lowest pheromone value on any edge incident to city $i \in N$
τ_{max}^i	largest pheromone value on any edge incident to city $i \in N$
w	number of ants that deposit pheromone in RANKBAS
ξ	pheromone evaporation level for the local pheromone update

Used Sets

M	set of all ants
N	set of all cities
N_i^k	set of cities that ant k has not yet visited when being at city i
S	set of all possible algorithm setups, resp. configurations
S_{ls}	set of algorithm setups $s \in S$ that use a local search method
$S_{\overline{ls}}$	set of algorithm setups $s \in S$ that do not use a local search method
T^k	tour generated by ant k (i.e. an ordered set of edges)

References

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* John Wiley & Sons, New York, USA, 1988.

[2] D. Anghinolfi, A. Boccalatte, M. Paolucci, and C. Vecchiola. Performance Evaluation of an Adaptive Ant Colony Optimization Applied to Single Machine Scheduling. In *SEAL '08: Proceedings of the 7th International Conference on Simulated Evolution and Learning*, pages 411–420, Berlin, Germany (a.o.), 2008. Springer-Verlag.

[3] J. Arabas, Z. Michalewicz, and J. J. Mulawka. GAVaPS – A Genetic Algorithm with Varying Population Size. In *ICEC '94: Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 73–78, Piscataway, USA, 1994. IEEE Press.

[4] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The Non-stochastic Multi-armed Bandit Problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[5] T. Bäck. Self-Adaptation in Genetic Algorithms. In *Proceedings of the 1st European Conference on Artificial Life*, pages 263–271, Cambridge, USA, 1992. MIT Press.

[6] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, Oxford, UK, 1996.

[7] T. Bäck, A. E. Eiben, and N. A. L. van der Vaart. An Empirical Study on GAs Without Parameters. In *PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 315–324, Berlin, Germany (a.o.), 2000. Springer-Verlag.

[8] P. Balaprakash, M. Birattari, and T. Stützle. Improvement Strategies for the F-Race Algorithm: Sampling Design and Iterative Refinement. In *HM '07: Proceedings of the 4th International Workshop on Hybrid Metaheuristics*, volume 4771 of *LNCS*, pages 108–122. Springer-Verlag, Berlin, Germany (a.o.), 2007.

[9] H.-G. Beyer and H.-P. Schwefel. Evolution Strategies – A comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[10] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A Racing Algorithm for Configuring Metaheuristics. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18, San Francisco, USA, 2002. Morgan Kaufmann Publishers.

[11] H. M. Botee and E. Bonabeau. Evolving Ant Colony Optimization. *Advanced Complex Systems*, 1:149–159, 1998.

[12] B. Bullnheimer, R. F. Hartl, and C. Strauss. A New Rank-Based Version of the Ant System: A Computational Study. *Central European Journal for Operations Research and Economics*, 7(1):25–38, 1999.

[13] G. Croes. A Method for Solving Traveling Salesman Problems. *Operations Research*, 6:791–812, 1958.

[14] L. Da Costa, A. Fialho, M. Schoenauer, and M. Sebag. Adaptive Operator Selection with Dynamic Multi-armed Bandits. In *GECCO '08: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 913–920, New York, USA, 2008. ACM Press.

[15] L. D. Davis and M. Mitchell. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, USA, 1991.

[16] K. De Jong. Parameter Setting in EAs: a 30 Year Perspective. In F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 1–18. Springer-Verlag, Berlin, Germany (a.o.), 2007.

[17] M. De la Maza and B. Tidor. Boltzmannn Weighted Selection Improves Performance of Genetic Algorithms. Technical Report A.I. Memo No. 1345, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, USA, 1991.

[18] M. de la Maza and B. Tidor. An Analysis of Selection Procedures with Particular Attention Paid to Proportional and Boltzmann Selection. In *ICGA V: Proceedings of the 5th International Conference on Genetic Algorithms*, pages 124–131, San Francisco, USA, 1993. Morgan Kaufmann Publishers.

[19] W. Domschke and A. Drexl. *Einführung in Operations Research.* Springer-Lehrbuch. Springer-Verlag, Berlin, Germany (a.o.), 7th edition, 2007.

[20] M. Dorigo, A. Colorni, and V. Maniezzo. The Ant System: An Autocatalytic Optimizing Process. Technical Report 91-016 Revised, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy, 1991.

[21] M. Dorigo and L. M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[22] M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press, Cambridge, USA, 2004.

[23] A. Dukkipati, Narasimha, and S. Bhatnagar. Cauchy Annealing Schedule: An Annealing Schedule for Boltzmann Selection Schemes in Evolutionary Algorithms. In *CEC '04: Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pages 55–62, Piscataway, USA, 2004. IEEE Press.

[24] A. Eiben, M. Schut, and A. Wilde. Boosting Genetic Algorithms with Self-Adaptive Selection. In *CEC '06: Proceedings of the IEEE Congress on Evolutionary Computation*, pages 477–482, Piscataway, USA, 2006. IEEE Press.

[25] A. Eiben and J. van der Hauw. Solving 3-SAT with Adaptive Genetic Algorithms. In *CEC '97: Proceedings of the IEEE Congress on Evolutionary Computation*, pages 81–86, Piscataway, USA, 1997. IEEE Press.

[26] A. E. Eiben, E. Marchiori, and V. A. Valkó. Evolutionary Algorithms with On-The-Fly Population Size Adjustment. In *PPSN VIII: Proceedings of the 8th International Conference on Parallel Problem Solving from Nature*, volume 3242 of *LNCS*, pages 41–50, Berlin, Germany (a.o.), 2004. Springer-Verlag.

[27] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. Parameter Control in Evolutionary Algorithms. In F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 19–46. Springer-Verlag, Berlin, Germany (a.o.), 2007.

[28] A. E. Eiben and Z. Ruttkay. Self-Adaptivity for Constraint Satisfaction: Learning Penalty Functions. In *CEC '96: Proceedings of the IEEE Congress on Evolutionary Computation*, pages 258–261, Piscataway, USA, 1996. IEEE Press.

[29] A. E. Eiben and J. van Hemert. SAW-ing EAs: Adapting the Fitness Function for Solving Constrained Problems. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 389–402. McGraw-Hill, New York, USA (a.o.), 1999.

[30] C. Fernandes, R. Tavares, C. Munteanu, and A. Rosa. Using Assortative Mating in Genetic Algorithms for Vector Quantization Problems. In *SAC '01: Proceedings of the 2001 ACM Symposium on Applied Computing*, pages 361–365, New York, USA, 2001. ACM Press.

[31] C. Fernandes, R. Tavares, and A. C. Rosa. niGAVaPS – Outbreeding in Genetic Algorithms. In *SAC '00: Proceedings of the 2000 ACM Symposium on Applied Computing*, pages 477–482, New York, USA, 2000. ACM Press.

[32] D. Gaertner and K. Clark. On Optimal Parameters for Ant Colony Optimization Algorithms. In *Proceedings of the International Conference on Artificial Intelligence*, pages 83–89, Las Vegas, USA, 2005. CSREA Press.

[33] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing, Reading, USA, 1989.

[34] D. E. Goldberg. Probability Matching, the Magnitude of Reinforcement, and Classifier System Bidding. *Machine Learning*, 5(4):407–425, 1990.

[35] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms. In *ICGA V: Proceedings of the 5th International Conference on Genetic Algorithms*, pages 56–64, San Francisco, USA, 1993. Morgan Kaufmann Publishers.

[36] D. E. Goldberg, K. Deb, and B. Korb. Messy Genetic Algorithms: Motivation, Analysis, and First Results. *Complex Systems*, 1(3):493–530, 1989.

[37] D. E. Goldberg, K. Deb, and D. Thierens. Toward a Better Understanding of Mixing in Genetic Algorithms. *Journal of the Society of Instrument and Control Engineers*, 32(1):10–16, 1993.

[38] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[39] N. Hansen and A. Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[40] N. Hansen, A. Ostermeier, and A. Gawelczyk. On the Adaptation of Arbitrary Normal Mutation Distributions in Evolution Strategies: The Generating Set Adaptation. In *ICGA VI: Proceedings of the 6th International Conference on Genetic Algorithms*, pages 57–64, San Francisco, USA, 1995. Morgan Kaufmann Publishers.

[41] J. Hesser and R. Männer. Towards an Optimal Mutation Probability for Genetic Algorithms. In *PPSN I: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 23–32, Berlin, Germany (a.o.), 1991. Springer-Verlag.

[42] R. Hinterding, Z. Michalewicz, and A. Eiben. Adaptation in Evolutionary Computation: A Survey. In *CEC '97: Proceedings of the IEEE Congress on Evolutionary Computation*, pages 65–69, Piscataway, USA, 1997. IEEE Press.

[43] R. Hinterding, Z. Michalewicz, and T. C. Peachey. Self-Adaptive Genetic Algorithm for Numeric Functions. In *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 420–429, Berlin, Germany (a.o.), 1996. Springer-Verlag.

[44] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, USA, 1975.

[45] H. H. Hoos and T. Stützle. *Stochastic Local Search : Foundations & Applications.* Morgan Kaufmann Publishers, San Francisco, USA, 2004.

[46] M. Khichane, P. Albert, and C. Solnon. An ACO-Based Reactive Framework for Ant Colony Optimization: First Experiments on the Constraint Satisfaction Problems. In *LION '09: Proceedings of the 3rd International Conference on Learning and Intelligent Optimization*, LNCS, pages 119–133, Berlin, Germany (a.o.), 2009. Springer-Verlag.

[47] O. Kovářík and M. Skrbek. Ant Colony Optimization with Castes. In *ICANN '08: Proceedings of the 18th International Conference on Artificial Neural Networks, Part I*, pages 435–442, Berlin, Germany (a.o.), 2008. Springer-Verlag.

[48] N. Krasnogor and J. Smith. A Memetic Algorithm With Self-Adaptive Local Search: TSP as a Case Study. In *GECCO '00: Proceedings of the Genetic and Evolutionary*

Computation Conference, pages 987–994, San Francisco, USA, 2000. Morgan Kaufmann Publishers.

[49] J. L. J. Laredo, C. Fernandes, J. J. Merelo, and C. Gagné. Improving Genetic Algorithms Performance via Deterministic Population Shrinkage. In *GECCO '09: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 819–826, New York, USA, 2009. ACM Press.

[50] J. Lis and M. Lis. Self-Adapting Parallel Genetic Algorithm with Dynamic Mutation Probability, Crossover Rate and Population Size. In *Proceedings of the 1st Polish National Conference on Evolutionary Computation*, pages 324–329, Warszava, Poland, 1996. Oficina Wydawnica Politechniki Warszawskiej.

[51] F. Lobo and G. Harik. A Parameter-Less Genetic Algorithm. In *GECOO '99: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 258–265, San Francisco, USA, 1999. Morgan Kaufmann Publishers.

[52] F. G. Lobo and C. F. Lima. Revisiting Evolutionary Algorithms with On-The-Fly Population Size Adjustment. In *GECCO '06: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1241–1248, New York, USA, 2006. ACM Press.

[53] F. G. Lobo and C. F. Lima. Adaptive Population Sizing Schemes in Genetic Algorithms. In F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 185–204. Springer-Verlag, Berlin, Germany (a.o.), 2007.

[54] S. Mahfoud. Boltzmann Selection. In *Handbook of Evolutionary Computation*, pages (C2.5:)1–4. Oxford University Press, Oxford, UK, 1997.

[55] T. Mahnig and H. Mühlenbein. A New Adaptive Boltzmann Selection Schedule SDS. In *CEC '01: Proceedings of the IEEE Congress on Evolutionary Computation*, pages 183–190, Piscataway, USA, 2001. IEEE Press.

[56] J. Paredis. The Symbiotic Evolution of Solutions and Their Representations. In *ICGA VI: Proceedings of the 6th International Conference on Genetic Algorithms*, pages 359–365, San Francisco, USA, 1995. Morgan Kaufmann Publishers.

[57] P. Pellegrini, D. Favaretto, and E. Moretti. On MAX - MIN Ant System's Parameters. In *ANTS '06: Proceedings of the 5th International Workshop on Ant Colony Optimization and Swarm Intelligence*, pages 203–214, Berlin, Germany (a.o.), 2006. Springer-Verlag.

[58] M. L. Pilat and T. White. Using Genetic Algorithms to Optimize ACS-TSP. In *ANTS '02: Proceedings of the 3rd International Workshop on Ant Colony Optimization and Swarm Intelligence*, pages 282–287, Berlin, Germany (a.o.), 2002. Springer-Verlag.

[59] M. Randall. Near Parameter Free Ant Colony Optimisation. In *ANTS '04: Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence*, pages 374–381, Berlin, Germany (a.o.), 2004. Springer-Verlag.

[60] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* PhD thesis, Technische Universität Berlin, Berlin, Germany, 1973.

[61] G. E. Robinson. Regulation of Division of Labor in Insect Societies. *Annual Review Entomology*, 37:637–665, 1992.

[62] R. Rosenberg. *Simulation of Genetic Populations with Biochemical Properties.* PhD thesis, University of Michigan, Ann Arbor, USA, Ann Arbor, USA, 1967.

[63] J. D. Schaffer and A. Morishima. An Adaptive Crossover Distribution Mechanism for Genetic Algorithms. In *ICGA II: Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 36–40, Hillsdale, USA, 1987. L. Erlbaum Associates.

[64] D. Schlierkamp-Voosen and H. Mühlenbein. Strategy Adaptation by Competing Subpopulations. In *PPSN III: Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, pages 199–208, Berlin, Germany (a.o.), 1994. Springer-Verlag.

[65] D. Schlierkamp-Voosen and H. Mühlenbein. Adaptation of Population Sizes by Competing Subpopulations. In *ICEC '96: Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 330–335, Piscataway, USA, 1996. IEEE Press.

[66] H.-P. Schwefel. *Numerical Optimization of Computer Models.* Wiley, Chichester, UK, 1981.

[67] C. G. Shaefer. The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique. In *ICGA II: Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 50–58, Hillsdale, USA, 1987. L. Erlbaum Associates.

[68] J. Smith. Parameter Perturbation Mechanisms in Binary Coded GAs with Self-Adaptive Mutation. In K. A. De Jong, R. Poli, and J. E. Rowe, editors, *Foundations of Genetic Algorithms 7*, pages 329–346. Morgan Kaufmann Publishers, San Francisco, USA, 2003.

[69] J. Smith and T. C. Fogarty. An Adaptive Poly-Parental Recombination Strategy. In *Selected Papers from AISB Workshop on Evolutionary Computing*, pages 48–61, Berlin, Germany (a.o.), 1995. Springer-Verlag.

[70] J. Smith and T. C. Fogarty. Recombination Strategy Adaptation via Evolution of Gene Linkage. In *ICEC '96: Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 826–831, Piscataway, USA, 1996. IEEE Press.

[71] R. E. Smith. Adaptively Resizing Populations: An Algorithm and Analysis. In *ICGA V: Proceedings of the 5th International Conference on Genetic Algorithms*, page 653, San Francisco, USA, 1993. Morgan Kaufmann Publishers.

[72] R. E. Smith and E. Smuda. Adaptively Resizing Populations: Algorithm, Analysis, and First Results. *Complex Systems*, 1(9):47–72, 1995.

[73] T. Stützle. *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications*, volume 220 of *DISKI*. Infix Publishers, Sankt Augustin, Germany, 1999.

[74] T. Stützle and H. H. Hoos. Improving the Ant System: A Detailed Report on the MAX-MIN Ant System. Technical Report AIDA–96–12, FG Intellektik, FB Informatik, Technische Universität Darmstadt, Darmstadt, Germany, 1996.

[75] T. Stützle and H. H. Hoos. MAX-MIN Ant System and Local Search for Combinatorial Optimization Problems. In *Meta-Heuristics: Advances and trends in local search paradigms for optimization*, pages 137–154. Kluwer Academic Publishers, Hingham, USA and Dordrecht, Netherlands, 1999.

[76] T. Stützle and H. H. Hoos. MAX-MIN Ant system. *Future Generation Computer Systems*, 16(9):889–914, 2000.

[77] D. Thierens. An Adaptive Pursuit Strategy for Allocating Operator Probabilities. In *GECCO '05: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1539–1546, New York, USA, 2005. ACM Press.

[78] P. Vajda, A. E. Eiben, and W. Hordijk. Parameter Control Methods for Selection Operators in Genetic Algorithms. In *PPSN X: Proceedings of the 10th International Conference on Parallel Problem Solving from Nature*, pages 620–630, Berlin, Germany (a.o.), 2008. Springer-Verlag.

[79] D. Whitley, K. Mathias, and P. Fitzhorn. Delta Coding: An Iterative Search Strategy for Genetic Algorithms. In *ICGA IV: Proceedings of the 4th International Conference on Genetic Algorithms*, pages 77–84, San Francisco, USA, 1991. Morgan Kaufmann Publishers.

[80] T.-L. Yu, K. Sastry, and D. E. Goldberg. Online Population Size Adjusting Using Noise and Substructural Measurements. In *CEC '05: Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2491–2498, Piscataway, USA, 2005. IEEE Press.

[81] C. Yunpeng, S. Xiaomin, and J. Peifa. Probabilistic Modeling for Continuous EDA with Boltzmann Selection and Kullback-Leibeler Divergence. In *GECCO '06: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 389–396, New York, USA, 2006. ACM Press.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Diplomarbeit ohne fremde Hilfe und nur unter Verwendung der zulässigen Mittel sowie der angegebenen Literatur angefertigt habe. Die Arbeit ist in gleicher oder ähnlicher Form im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Darmstadt, den 20.12.2009