Chapter 8
Parameter Adaptation in Ant Colony Optimization

Thomas Stützle, Manuel López-Ibáñez, Paola Pellegrini, Michael Maur, Marco Montes de Oca, Mauro Birattari, and Marco Dorigo

8.1 Introduction

Ant colony optimization (ACO) is a metaheuristic inspired by the foraging behavior of ants [13, 9, 14, 11, 15, 8]. In ACO algorithms, artificial ants are probabilistic solution construction procedures that are biased by artificial pheromones and heuristic information. Heuristic information can be derived from a problem instance to guide ants in the solution construction process. Pheromones are represented as numerical information that is modified iteratively to reflect the algorithm's search experience. Modifications bias the search towards good quality solutions [45].

The behavior of ACO algorithms depends strongly on the values given to parameters [11, 18]. In most ACO applications, parameter values are kept constant throughout each run of the algorithm. However, varying the parameters at computation time, either in prescheduled ways or depending on the search progress, can enhance the performance of an algorithm. Parameter control and parameter adaptation are recurring themes in the field of evolutionary algorithms (EAs) [31]. The adaptation of parameters while solving a problem instance by exploiting machine learning techniques is also the unifying theme in the research area of reactive search [3]. In the ACO literature as well, several strategies have been proposed and tested for modifying parameters while solving a problem instance.

Thomas Stützle, Manuel López-Ibáñez, Marco Montes de Oca, Mauro Birattari, Marco Dorigo
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
e-mail: {stuetzle,manuel.lopez-ibanez,mmontes,mbiro,mdorigo}@ulb.ac.be

Michael Maur
Fachbereich Rechts- und Wirtschaftswissenschaften, TU Darmstadt, Darmstadt, Germany
e-mail: maur@stud.tu-darmstadt.de

Paola Pellegrini
Dipartimento di Matematica Applicata, Università Ca' Foscari Venezia, Venezia, Italia
e-mail: paolap@unive.it

Y. Hamadi et al. (eds.), *Autonomous Search*,
DOI 10.1007/978-3-642-21434-9_8,

In this chapter, we first review available research results on parameter adaptation in ACO algorithms. We follow the three classes of parameter control strategies discussed by Eiben et al. [17]. Then, we analyze the development of the solution quality reached by ACO algorithms over computation time for specific parameter values. The goal is to identify situations where the best fixed parameter settings depend strongly on the available computation time because it is exactly in such situations that prescheduled variation of parameter values can improve strongly the anytime performance [44] of an algorithm. We observe strong dependencies for the MAX-MIN Ant System (MMAS) [42], and we show that prescheduled parameter variation actually leads to much improved behavior of MMAS over computation time without compromising the final solution quality obtained.

This chapter is structured as follows. In Section 8.2 we give an introductory description of the main ACO variants, which are also mentioned later in the chapter. After a short review of parameter adaptation in Section 8.3, we discuss relevant literature concerning ACO in Section 8.4. The experimental part in this chapter is divided into two sections: Section 8.5 describes experiments with fixed parameter settings, whereas Section 8.6 describes experiments with prescheduled parameter variation. From the review of the literature and our experimental study, we provide some conclusions and suggestions for future research in Section 8.7.

8.2 Ant Colony Optimization

The earliest ACO algorithms used the traveling salesman problem (TSP) as an example application. The TSP is typically represented by a graph $G = (V, E)$, V being the set of $n = |V|$ vertices, representing the cities, and E being the set of edges that fully connect the vertices. A distance d_{ij} is associated with each edge (i, j). The objective is to find a Hamiltonian cycle of minimum total cost. The TSP is a computationally hard problem, but the application of ACO algorithms to it is simple, which explains why ACO applications to the TSP have played a central role in the development of this algorithmic technique.

8.2.1 Ant Colony Optimization for the TSP

When applying ACO to the TSP, a pheromone value τ_{ij} is associated with each edge $(i, j) \in E$. The pheromone value represents the attractiveness of a specific edge for the ants, according to the experience gained at runtime: the higher the amount of pheromone on an edge, the higher the probability that ants choose it when constructing solutions. Pheromone values are iteratively updated by two mechanisms: pheromone evaporation and pheromone deposit. In addition to the pheromone trails, the ants' solution construction process is also biased by a heuristic value $\eta_{ij} = 1/d_{ij}$,

```
procedure ACOMetaheuristic
    ScheduleActivities
        ConstructSolutions
        DaemonActions    //optional
        UpdatePheromones
    end-ScheduleActivities
end-procedure
```

Fig. 8.1: ACO metaheuristic for NP-hard problems in pseudo-code

which represents the attractiveness of each edge (i, j) from a greedy point of view.

The algorithmic outline of an ACO algorithm (see Figure 8.1) contains three main procedures: ConstructSolutions, DaemonActions, and UpdatePheromones. The main characteristics of these procedures are as follows.

- ConstructSolutions. This procedure includes the routines needed for ants to construct solutions incrementally. After the selection of the starting city, one node at a time is added to an ant's path. An ant's decision about where to go next is biased by the pheromone trails τ_{ij} and the heuristic information η_{ij}. In general, the higher the two values, the higher the probability of choosing the associated edge. Typically, two parameters, $\alpha > 0$ and $\beta \geq 0$, are used to weigh the relative influence of the pheromone and the heuristic values, respectively. The rule that defines the ant's choice is specific to each ACO variant.
- DaemonActions. This procedure comprises all problem-specific operations that may be considered for boosting the performance of ACO algorithms. The main example of such operations is the introduction of a local search phase. In addition, daemon actions implement centralized tasks that cannot be performed by an individual ant. This type of procedures is for optional use, but typically several daemon actions are very useful for significantly improving the performance of ACO algorithms [11].
- UpdatePheromones. This procedure updates the pheromone trail values in two phases. First, pheromone evaporation is applied to decrease pheromone values. The degree of decrement depends on a parameter $\rho \in [0, 1]$, called *evaporation rate*. The aim of pheromone evaporation is to avoid an unlimited increase of pheromone values and to allow the ant colony to forget poor choices made previously. A pheromone deposit is then applied to increase the pheromone values that belong to good solutions the ants have generated. The amount of pheromone deposited and the solutions considered are peculiar to each ACO variant.

ACO algorithms involve a number of parameters that need to be set appropriately. Of these, we already have mentioned α and β, which are used to weigh the relative influence of the pheromone and heuristic values in the ants' solution construction. The role of these parameters in biasing the ants' search is intuitively similar. Higher values of α emphasize differences in the pheromone values, and higher

values of β have the same effect on the heuristic values. The initial value of the pheromones, τ_0, has a significant influence on the convergence speed of the algorithm; however, its recommended setting depends on the particular ACO algorithm. The evaporation rate parameter, ρ, $0 \leq \rho \leq 1$, regulates the degree of the decrease in pheromone trails. If ρ is low, the influence of the pheromone values will persist longer, while high values of ρ allow fast forgetting of previously very attractive choices and, hence, allow faster focus on new information that incorporate into the pheromone matrix. Another parameter is the number of ants in the colony, m. For a given computational budget, such as maximum computation time, the number of ants is a critical parameter for determining the trade-off between the number of iterations that can be made and how broad the search is at each of the iterations. In fact, the larger the number of ants per iteration, the fewer the iterations of the ACO algorithm.

8.2.2 ACO Variants

The ACO framework can be implemented in many different ways. In the literature, several ACO algorithms have been proposed, which differ in some choices characterizing the ConstructSolutions and UpdatePheromones procedures. Three of the main variants are described next. For a description of other variants we refer the interested reader to Dorigo and Stützle [11].

8.2.3 Ant System

Ant System (AS) is the first ACO algorithm proposed in the literature [12, 13]. In AS, an ant k in node i chooses the next node j with probability given by the random proportional rule, defined as

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{h \in N^k} [\tau_{ih}]^{\alpha} \cdot [\eta_{ih}]^{\beta}}, \tag{8.1}$$

where N^k is its feasible neighborhood. The feasible neighborhood excludes nodes already visited in the partial tour of ant k, and it may be further restricted to a candidate set of the nearest neighbors of a city i. Once an ant has visited all nodes, it returns to its starting node.

During the execution of the UpdatePheromones procedure in AS, all m ants deposit pheromones at each iteration. The pheromone trail values are updated as

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta \tau_{ij}^{k}, \tag{8.2}$$

where $\Delta\tau_{ij}^k$ is defined as

$$\Delta\tau_{ij}^k = \begin{cases} F(k) & \text{if edge } (i,j) \text{ is part of the solution constructed by ant } k, \\ 0 & \text{otherwise,} \end{cases} \tag{8.3}$$

where $F(k)$ is the amount of pheromone deposited on the edges of the solution constructed by ant k. $F(k)$ is equal to the reciprocal of the cost of the solution constructed by ant k, possibly multiplied by a constant Q. Hence, the better the solution, the higher the amount of pheromone deposited by an ant.

8.2.4 MAX–MIN Ant System

MAX–MIN Ant System (MMAS) is an improvement over the AS algorithm [42]. The main difference is the handling of the pheromone trails update. Firstly, only one solution is used for the pheromone deposit. This is typically either the iteration-best solution or the best-so-far solution, that is, the best solution since the start of the algorithm. Secondly, all pheromone values are bounded by the interval $[\tau_{\min}, \tau_{\max}]$. The pheromone update rule used is

$$\tau_{ij} \leftarrow \max\{\tau_{\min}, \min\{\tau_{\max}, (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{\text{best}}\}\} \tag{8.4}$$

where $\Delta\tau_{ij}^{\text{best}}$ is defined as

$$\Delta\tau_{ij}^{\text{best}} = \begin{cases} F(s^{\text{best}}) & \text{if edge } (i,j) \text{ is part of the best solution } s^{\text{best}}, \\ 0 & \text{otherwise.} \end{cases} \tag{8.5}$$

$F(s^{\text{best}})$ is the reciprocal of the cost of the solution considered for the deposit. For more details on the pheromone initialization and the use of occasional pheromone trail reinitializations, we refer the reader to Stützle and Hoos [42].

8.2.5 Ant Colony System

Ant Colony System (ACS) [10] differs in several ways from AS and MMAS. ACS uses the *pseudorandom proportional rule* in the solution construction: with a probability q_0 the next city to visit is chosen as

$$j = \arg\max_{h \in N^k} \{\tau_{ih} \cdot \eta_{ih}^{\beta}\}, \tag{8.6}$$

that is, the most attractive edge is selected greedily with fixed probability. With probability $1-q_0$, the AS random proportional rule defined by Equation 8.1 is used. In ACS, the parameter α is fixed to 1, and, therefore, it is often omitted.

The pheromone deposit of ACS modifies only the pheromone values of edges from one solution. As in MMAS, this solution is either the iteration-best or the best-so-far solution. The ACS pheromone update formula is

$$\tau_{ij} \leftarrow \begin{cases} (1-\rho)\cdot\tau_{ij} + \rho\cdot\Delta\tau_{ij} & \text{if } (i,j) \text{ is part of the best solution } s^{\text{best}}, \\ \tau_{ij} & \text{otherwise,} \end{cases} \tag{8.7}$$

with $\Delta\tau_{ij} = F(s^{\text{best}})$.

A local pheromone update rule is applied during the solution construction of the ants. Each time an ant traverses an edge (i,j), τ_{ij} is modified as

$$\tau_{ij} \leftarrow (1-\xi)\cdot\tau_{ij} + \xi\cdot\tau_0, \tag{8.8}$$

where $\xi \in (0,1)$ is a parameter called pheromone decay coefficient, and τ_0 is the initial value of the pheromones. In ACS, τ_0 is a very small constant with value $1/(n\cdot L_{nn})$, where L_{nn} is the length of a nearest neighbor tour. The local pheromone update aims at avoiding stagnation: it decreases the pheromone value on the previously used edges and makes them less attractive for other ants.

8.3 Overview of Parameter Adaptation Approaches

The dependency of the performance of metaheuristics on the settings of their parameters is well known. In fact, finding appropriate settings for an algorithm's parameters is considered to be a nontrivial task and a significant amount of work has been devoted to it. The approaches for tackling this task can roughly be divided into offline versus online procedures.

Offline tuning has the goal of finding appropriate settings for an algorithm's parameters before the algorithm is actually deployed. Traditionally, offline tuning has mostly been done in a trial-and-error fashion. This process is time-consuming, human-intensive and error-prone, and it often leads to the uneven tuning of different algorithms. Moreover, tuning by trial and error depends much on the intuition and experience of the algorithm developer, and it is typically undocumented and therefore not reproducible. More recently, increasing effort has been devoted to methods that allow for search-based, hands-off tuning of algorithm parameters. The study of techniques for automatic algorithm configuration is currently a rather active research area. Still, these methods typically come up with one specific parameter setting which then remains the same while the algorithm solves a particular instance.

An alternative to offline tuning is online tuning. Typically, this consists of the modification of an algorithm's parameter settings while solving a problem instance. A potential advantage of online modification of parameters is that algorithms may

adapt better to the particular instance's characteristics. When instances are relatively heterogeneous, parameter settings that result in good performance on average across all instances may lead to much worse results on some instances. Online parameter adaptation may also be useful for achieving the best performance for a stage of search. It is often possible to identify an algorithm's explorative and exploitative search phases, and good parameter settings in these phases may again be very different. Finally, if algorithms are applied in situations that are very different from the context in which they have been developed or tuned offline, allowing parameters to change online may increase an algorithm's robustness.

There are different ways of modifying parameters during the run of an algorithm. Many strategies have been widely studied in the context of EAs, and Eiben et al. [17] give a possible classification of these strategies. Perhaps the simplest way is to define the parameter variation rule before actually running the algorithm. In such an approach, the problem is observed from an offline perspective: Static parameters are substituted with (deterministic or randomized) functions depending on the computational time or the number of algorithm iterations. Eiben et al. [17] call such strategies *deterministic* parameter control; however, we prefer the term *prescheduled* parameter variation, because the adjective "deterministic" does not correctly characterize this way of controlling parameters, given that the schedule could also allow randomized choices. Even if prescheduled parameter variation is an online tuning method, it does not make the offline tuning problem disappear since also the parameters that define the schedule need to be appropriately set.

An alternative is to use *adaptive parameter settings*, where the parameter modification scheme is defined as a function of some statistics on the algorithm behavior. Various measures can be used for this online adaptation. They can be grouped depending on whether they are based on absolute or relative evidence. In the first case, the adaptation strategy monitors the occurrence of some events during the run, for example, some fixed threshold of the distance between the solutions visited. Surpassing the threshold then triggers a set of rules for parameter variation. In the second case, the adaptation strategy considers the relative difference between the performance achieved with different parameter settings, and adapts the parameter values to resemble the most successful ones. For parameter adaptation to work, some additional decisions need to be made beyond the ones strictly related to the implementation of the algorithm. In particular, the equations that describe the parameter update need to be defined a priori and it is hoped that the mechanisms used are very robust with respect to their definitions.

A further possibility that has been the object of studies consists of having the parameters modified at run time by the algorithm itself. Specifically, dimensions that represent parameters of exploration strategies are added to the search space of the problem. The optimization process is then executed in this new space. Eiben et al. [17] name this approach *self-adaptation*. Taking the notion of self-adaptation a step further, we call *search-based adaptation* strategies that use a search algorithm different from the underlying algorithm for parameter adaptation. This class of strategies includes techniques such as local search and EAs for adapting the parameters of ACO algorithms.

Table 8.1: Schematic description of the literature on adaptive ACO. Some of the articles propose general adaptation strategies that could be used for several parameters. Here we only indicate the parameters that have been adapted experimentally

Authors	Adaptation strategy	ACO variant	Parameters
Merkle and Middendorf [34]	pre-scheduled	variant of AS	β
Merkle et al. [35]	pre-scheduled	variant of AS	β, ρ
Meyer [36]	pre-scheduled	AS	α
Randall and Montgomery [40]	adaptive	ACS	candidate set
Chusanapiputt et al. [6]	adaptive	AS	α, β
Li and Li [28]	adaptive	new variant	α, β
Hao et al. [24]	adaptive	ACS	ρ
Kovářík and Skrbek [27]	adaptive	variant of MMAS	β
Li et al. [29]	adaptive	variant of ACS	q_0, pheromone update
Cai et al. [5]	adaptive	ACS	ρ
Randall [39]	self-adaptation	ACS	β, ρ, q_0, ξ
Martens et al. [32]	self-adaptation	MMAS	α, β
Förster et al. [19]	self-adaptation	new variant	pheromone update
Khichane et al. [26]	self-adaptation	MMAS	α, β
Pilat and White [38]	search-based adaptation	ACS	β, ξ, q_0
Gaertner and Clark [20]	search-based adaptation	AS–ACS combination	β, ρ, q_0
Hao et al. [23]	search-based adaptation	variant of ACS	β, ρ, q_0
Garro et al. [22]	search-based adaptation	variant of ACS	algorithm specific
Ling and Luo [30]	search-based adaptation	variant of ACS	α, ρ, Q
Amir et al. [1]	search-based adaptation	ACS	β, q_0
Anghinolfi et al. [2]	search-based adaptation	variant of ACS	β, q_0
Melo et al. [33]	search-based adaptation	multi-colony ACS	α, β, ρ, q_0

8.4 Parameter Adaptation in ACO

The study of the impact of various parameters on the behavior of ACO algorithms has been an important subject since the first articles [12, 13]. We schematically summarize in Table 8.1 the main approaches that have been used in the ACO literature to adapt parameter values, following roughly the classes defined by Eiben et al. [17]. This summary shows that the most frequently chosen parameters for adaptation are α, β, q_0 (in the case of ACS), and those that control the pheromone update. We describe these approaches in the following sections.

8.4.1 Prescheduled Variation of Parameter Settings

There is surprisingly little work on prescheduled parameter variation for ACO algorithms. Merkle and Middendorf [34] describe a contribution that considers an ACO algorithm for the resource-constrained project scheduling problem. Its authors propose decreasing the value of the parameter β linearly over the run of an algorithm from an initial value of 2 to 0. In a subsequent study, Merkle et al. [35] consider the same problem and its authors propose modifying the parameter β and the evapora-

tion rate ρ. For β they propose a schedule as described before. For ρ, they propose starting with a small value to increase the initial exploration of the search space, and later setting the evaporation rate to a high value for an intensive search around the best solutions found by the algorithm.

Meyer [36] proposes a variant of AS called α-annealing. The idea at the basis of its author's work is to change the value of α according to some annealing schedule. Increasing α slowly throughout the search can keep diversity in the beginning and gradually increase the selective pressure to cover better regions of the search space in the later phases.

8.4.2 Adaptive Approaches

Many of the approaches proposed in the literature can be classified as adaptive. In these approaches, some parameters are modified according to some rules that take into account the search behavior of the ACO algorithm. The average λ-branching factor [21] is one of the first proposed measures of ACO behavior. Other measures include entropy-based measures for the pheromone, dispersion of solutions generated by the algorithm, or simply the quality of the solutions generated [7, 37]. Favaretto et al. [18] propose a technique for measuring the effect of parameter variation on the exploration performed; this technique may also serve as an indicator for defining parameter adaptation strategies.

In an early discussion of the usefulness of parameter adaptation in ACO algorithms, Merkle and Middendorf [34] propose a decomposition of the search into different phases to allow for the development of parameter adaptation strategies. Several strategies have been proposed later and we divide these in two groups: adaptations based on the dispersion of the pheromone trails, and adaptations based on the quality of solutions. Within the first group, Li and Li [28] introduce an ACO algorithm that varies the parameters α and β over time. Their parameter adaptation strategy considers a measure of the entropy on the action choice probabilities of the ants during solution construction and they aim at obtaining a schedule for α and β. During the early stage of the search, the value of α is small enough to allow extensive exploration of the search space; the value of α increases over time to improve the local search ability of the algorithm. They suggest the opposite schedule for β. Li et al. [29] propose a variant of ACS that uses a "cloud-based fuzzy strategy" for choosing the solution to be used in the global pheromone update. The main idea is that, as the pheromones get more concentrated around a single solution, tours other than the best-so-far one have a good chance of depositing pheromone. Additionally, they adapt the parameter q_0 with the goal of decreasing it as soon as the pheromone trails concentrate on very few edges. Chusanapiputt et al. [6] propose a variation of AS for dealing with the unit commitment problem. Three of the algorithm's parameters are adapted based on pheromone dispersion.

In a second group of papers, the driver of adaptation is the quality of the solutions generated. Hao et al. [24] and Cai et al. [5] propose a variant of ACS for the TSP. In their implementation, a different value of the parameter ρ is associated with each ant depending on the quality of its solution. This mechanism aims at having high-quality solutions contribute more pheromone than low-quality ones. Amir et al. [1] add a fuzzy logic controller module to the ACS algorithm for the TSP for adapting the value of β and q_0. The adaptive strategy uses two performance measures: the difference between the optimal solution (or the best known solution) and the best one found and the variance of the solutions visited by a population of ants. Kovářík and Skrbek [27] describe an approach that divides the ant colony into groups of ants using different parameter settings. They adapt the number of ants in each of the groups depending on the improvement of the solution quality obtained by each group; however, they do not give all details of the adaptation strategy. An analysis of the experimental results of adapting the values of β indicates that better initial performance is obtained with high values of β while towards the end of the run low values of β are preferable. Randall and Montgomery [40] apply an adaptation mechanism to an ACS algorithm that uses a *candidate set strategy* as a speedup procedure. At each step, the component to be added to the partial solution under construction is chosen from the ones belonging to such a set. The composition of the set is modified throughout the run. Elements that give low probability values are eliminated temporarily from the search process (they become tabu). After a number of iterations, they are added again to the candidate set. The threshold for establishing which elements are tabu is varied throughout the search process, depending on the quality of solutions being produced.

8.4.3 Search-Based Parameter Adaptation

Various adaptive ACO strategies fall into the category of self-adaptive strategies [17], where an algorithm tunes itself by integrating its parameters into its search task. We first present strategies that are "purely self-adaptive" in the original meaning used by Eiben et al. [17]. Later, we discuss approaches that use other search algorithms than ACO for adapting parameters. Given that these strategies are search-based, most of the approaches discussed in the following use as feedback the quality of the solutions generated.

8.4.3.1 Pure Self-Adaptive Approaches

The first self-adaptive approach to ACO is by Randall [39]. He suggests discretizing the parameter range and associating with each resulting value of a parameter a pheromone trail that gives the desirability of choosing it. In his approach, each ant adapts its own parameter settings and chooses them at each iteration before solution

construction. This mechanism is tested by adapting the parameters β, q_0, ρ, and ξ for ACS applied to the TSP and the quadratic assignment problem. The comparison of the results to the default parameter settings is somehow inconclusive.

Martens et al. [32] propose a self-adaptive implementation of MMAS applied to the generation of decision rule-based classifiers. In their AntMiner+ algorithm, ants choose suitable values for the parameters α and β. This is done by introducing for each parameter a new vertex group in the construction graph. The values of α and β are limited to integers between 1 and 3. Unlike in the previous paper, here parameters are treated as interdependent.

Förster et al. [19] apply the same idea to an ACO approach for a multi-objective problem. The parameters adapted are specific to the algorithm proposed, but they are mostly related to pheromone deposit. As in Randall [39], the dependence among parameters is neglected, that is, no new nodes are added to the construction graph. A separate pheromone matrix is recorded, each column representing a parameter to be adapted. Before starting the solution construction, each ant selects probabilistically its own parameter settings based on the pheromone matrix.

Khichane et al. [26] study a self-adaptive mechanism to tune the parameters α and β of their implementation of MMAS and apply it to constraint satisfaction problems. However, differently from the previous works, they do not define parameter settings at the level of an individual ant. For each iteration one common parameter setting for the whole ant colony is defined. The two parameters are considered independent of each other. The authors propose two variants of the parameter adaptation mechanism. In the first one, called global parameter learning ant-solver (GPL-Ant-solver), the colony uses the same parameter setting during the solution construction of each ant. In the second one, called distributed parameter learning ant-solver (DPL-Ant-solver), at each step of the solution construction the colony chooses new values for α and β; hence, in this case the pheromones that encode specific parameter settings refer to the desirability of choosing a specific parameter value for a specific construction step. In an experimental evaluation of the two variants, both are shown to reach similar performance levels. A comparison with an offline tuned version of their ant-solver shows that for some instances the adaptive version performs better while for others the opposite is true.

8.4.3.2 Other Search Algorithms for Adapting Parameters

Pilat and White [38] test two ways of using an EA to adjust parameters of an ACS algorithm; one of them has to do online tuning. Their approach to online tuning uses an EA to determine, at each ACS iteration, the parameter settings of four ants before constructing solutions. The EA in turn uses the constructed solutions to further evolve a set of good parameter settings. The authors choose three parameters for adaptation, namely, β, q_0, and ξ. Their results are somewhat inconclusive. This approach is similar to the mechanism used in an earlier paper by White et al. [43], where the authors evolve the parameters α and β in an ACO algorithm for a telecommunications routing problem. As an alternative to the online tuning of ACO

parameters by an EA, Pilat and White [38] explore the use of an EA as an offline tuning mechanism, analogously to Botee and Bonabeau [4].

Gaertner and Clark [20] propose a similar adaptive approach. In their work, every ant is initialized with a random parameter combination, where the parameter values are chosen from a predefined range. Over time, the entire population of ants evolves, breeding ants with parameter combinations which find improved solutions. In their experiments, the authors consider an algorithm based on a combination of AS and ACS for the TSP. They test their approach on three parameters: β, ρ and q_0.

Hao et al. [23] propose a variant of ACS in which each ant is characterized by its own parameter setting. The usual random-proportional rule is applied for selecting subsequent moves. After each iteration, the parameter configurations are modified using a particle swarm optimization (PSO) approach. Three parameters are adapted throughout the algorithm's run, namely β, ρ and q_0. If the PSO mechanism assigns a value outside a predefined range to a parameter, then the parameter is randomly reinitialized. Following a similar idea, Ling and Luo [30] propose using an artificial fish swarm algorithm for exploring the parameter space. Its authors also consider a variant of ACS and vary the three parameters α, ρ and Q, a parameter that influences the amount of pheromone an ant deposits. The main difference between this work and the one of Hao et al. [23] is that Ling and Luo use the same parameter setting for all ants.

Garro et al. [22] present an algorithm that evolves parameters using an EA. An individual in the EA represents an ant characterized by specific parameter values. The authors study a variant of ACS for automatically determining the path a robot should follow from its initial position to its goal position. They adapt three parameters of a newly proposed state transition rule.

Anghinolfi et al. [2] adapt two parameters using a local search in the parameter space. They define the neighborhood of the current parameter setting as all possible combinations of parameter settings that can be obtained by increasing or decreasing each parameter value by a fixed amount. Therefore, in the case of two parameters, at each iteration five parameter configurations are tested: the incumbent one and its four resulting neighbors. The test is done by assigning each parameter combination to one of five equal-sized groups of ants; each group then uses its parameters to generate solutions. After each iteration, the incumbent parameter setting is changed to the one that produced the iteration-best solution. In their experiments, the authors adapt two parameters of a variant of ACS, namely β and q_0. They observe better performance of the adaptive strategy than a tuned, fixed parameter setting.

Finally, Melo et al. [33] propose a multi-colony ACS algorithm, where several colonies of ants try to solve the same problem simultaneously. Each colony uses different parameter settings for α, β, ρ and q_0. Apart from exchanging solutions among the colonies, their proposal includes a mutation operator that replaces the parameter settings of the worst colony with the value of the same parameter in the best colony modified by a small, uniformly random value.

8.4.4 Conclusions from the Review

The review above shows that there is ongoing interest in parameter adaptation in the ACO literature. However, we also observe that several of the contributions apply adaptive techniques without prior in-depth understanding of the effect of individual parameters. Without such an understanding, decisions about which parameters to adapt and how to adapt them are mostly arbitrary. In particular, we did not find in our review any systematic study of the effect of different parameter settings on the anytime behavior of ACO algorithms. It is our intuition that such an analysis can inform decisions not only about which parameters may be worth varying during runtime, but also about how to perform such variations. Moreover, the anytime behavior of fixed parameter settings provides a baseline for evaluating the performance of parameter adaptations. In the following sections, we first provide a systematic study of the anytime behavior of ACO algorithms, and we use the knowledge acquired from it to design successful schemes for prescheduled parameter variation.

8.5 Experimental Investigation of Fixed Parameter Settings

In this section, we examine the effect that various parameters have on the performance of ACS and MMAS. In particular, we are interested in the development of the best-so-far solution over time when varying one parameter at a time. Our goal is to identify which parameter settings produce the best results at any moment during the run of the algorithm. Clearly, a parameter setting that produces very good solutions during the initial stages of a run but leads to much worse results later is an interesting candidate for having its settings varied online. In other words, we are interested in the anytime behavior [44] of specific parameter settings to clearly identify opportunities for the adaptation of parameter values over the computation period.

Our experimental analysis is based on the publicly available ACOTSP software [41], which we compiled with `gcc`, version 3.4. Experiments are carried out on a cluster of Intel Xeon™ E5410 quad-core processors running at 2.33 GHz with 6 MB L2 Cache and 8 GB RAM under Rocks Cluster GNU/Linux. Due to the sequential implementation of the code, only one core is used for running the executable.

We test two ACO algorithms, ACS and MMAS. Table 8.2 gives the default values for the parameters under study. In each experiment where one parameter is varied, the others are kept fixed at their default values. For the remaining parameters, that is, τ_0, ξ (for ACS), the choice of iteration-best or best-so-far update, and so on, we use the default values given by Dorigo and Stützle [11], which are also the default values of the ACOTSP package. We test the algorithms with and without the use of the first-improvement 2-opt local search provided by the ACOTSP package. For the experiments, TSP instances are randomly generated using the instance generator provided for the 8th DIMACS challenge on the TSP; in particular, points are generated uniformly at random in a square of side length 1,000,000. When using ACO

Table 8.2: Default settings of the parameters under study for MMAS and ACS without local search and with 2-opt local search

Algorithm	β	ρ	m	q_0
ACS	2	0.1	10	0.9
ACS + 2-opt	2	0.1	10	0.98
MMAS	2	0.02	n	0.00
MMAS + 2-opt	2	0.2	25	0.00

algorithms without local search, the tests are done on instances of size 100, 200, 400 and 800; because of the much higher performance of the ACO algorithms when local search is used, we use with local search larger instances of size 1,500, 3,000 and 6,000 to minimize possible floor effects. The presentation of the experimental results is based on the development of the relative deviation of the best solution found by an ACO algorithm from the optimal solution (or the best known solution for the instance of size 6,000). Each of the curves of the solution quality over time, or SQT curves [25], is the average of 25 executions of each parameter setting. Since we only present plots, we give for each setting results on only one instance. However, the trends are the same on all instances and, hence, the plots are representative of the general results.

8.5.1 Fixed Parameter Settings for Ant Colony System

In the case of ACS, we study the effect of β, which regulates the influence of the heuristic information; m, the number of ants; ρ, the evaporation factor; and q_0, the probability of making a deterministic choice in Equation 8.6. Here, we present SQT curves only for the case where ants' solutions are improved by a local search for the instance of size 3,000. The final conclusions concerning the usefulness of the variation of parameters at run-time were the same on the other instances and when using ACS without local search. Figures 8.2 to 8.5 report the results on parameters β, m, ρ, and q_0, respectively.

The main overall conclusion we obtain from these results is that very often there is a single parameter setting that performs best during most of the available runtime. Hence, there does not appear to be a clear benefit to varying the parameter settings at run-time. This conclusion remains the same if ACS is run without local search; the main difference is that the performance is more variable and more dependent on specific parameter values. In more detail, the observations and conclusions that arise for the single parameters from the presented results are the following.

β, *Figure 8.2:* Medium range values of β equal to 2 or 5 produce the best results during most of the runtime. Smaller values of β are initially worse but,

after enough computation time, eventually match the results of the default value. Much larger values (e.g., $\beta = 10$) are quickly overshadowed by smaller ones.

m, Figure 8.3: The default value of ten ants results in very good anytime performance. Interestingly, very small values (notably $m = 1$) make the algorithm perform slightly worse during the whole runtime, whereas much larger values ($m = 100$) lead to much worse results. The latter effect is probably due to too much diversification because of the application of the local pheromone update rule in ACS.

ρ, *Figure 8.4:* Surprisingly, the differences among different settings of ρ are almost imperceptible. Without local search (not shown here), large ρ values produce faster convergence. However, after a short time small values close to the default ($\rho = 0.1$) produce progressively better results.

q_0, *Figure 8.5:* As suggested in the literature, good values of q_0 tend to be close to 1. In extreme cases, a value of 1 quickly leads to search stagnation, while values smaller than 0.75 produce very slow convergence towards good solutions. Similar results are obtained when local search is disabled.

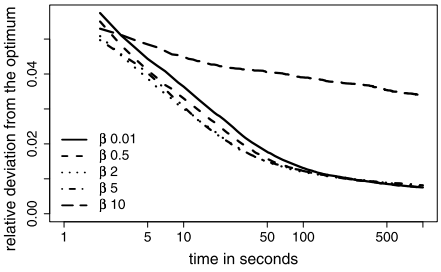

Fig. 8.2: ACS with various values of β

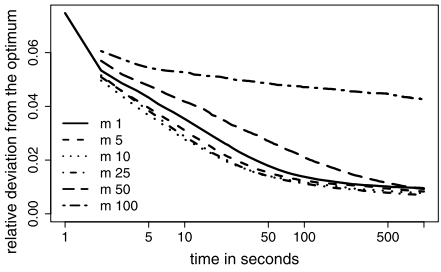

Fig. 8.3: ACS with various numbers of ants (m)

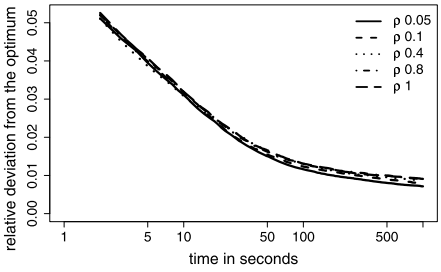

Fig. 8.4: ACS with various values of ρ

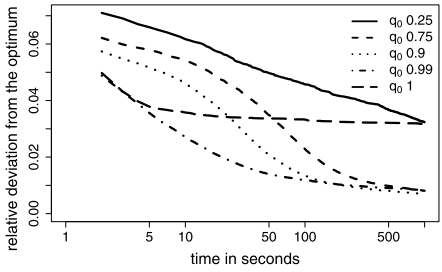

Fig. 8.5: ACS with various values of q_0

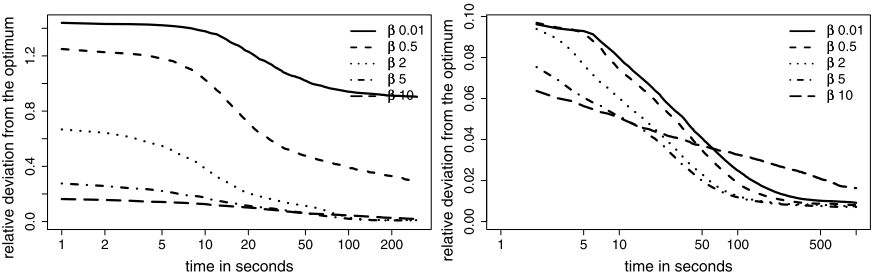

Fig. 8.6: MMAS with various fixed values of β; left plot without local search and right plot with local search

8.5.2 *Fixed Parameter Settings for MAX–MIN Ant System*

We now study the impact of the same parameters, β, m, ρ, and q_0, on the anytime behavior of MMAS. For MMAS, the behavior is more interesting from a parameter adaptation point of view. We therefore present results for the cases with and without local search. Results without local search are for one instance with 400 nodes, whereas with local search, they are for an instance with 3,000 nodes.

β, Figure 8.6: Without local search (upper part of Figure 8.6), MMAS requires relatively large values of β, which produce a significant advantage, especially during the initial part of the run, over the default setting of $\beta = 2$. While results with the default setting eventually match the results obtained with higher settings, values of β less than 2 lead to quite poor performance. With local search, the differences are much smaller and a setting of $\beta = 10$ is quickly overshadowed by lower ones. This suggests that starting with a high value of β may enhance the performance of MMAS at the beginning of the run, but a value close to the default may produce better results for larger computation times.

m, Figure 8.7: With and without local search, the number of ants shows a clear trade-off between early and late performance. In particular, a low number of ants (for example, $m = 5$) produces the best results during the early stages of the algorithm run. However, a higher number of ants (for example, $m = 100$) obtains much better results towards the end of the run. Without local search, the fast initial progress with few ants soon levels off and apparently leads to search stagnation. In this case, the default setting of $m = 400$ appears to be already too high, and it slows down the algorithm compared to using 100 ants without improving the final result. With local search, the SQT curves cross for the different parameter settings and those with few ants ($m = 1$ and $m = 5$) result in worse final solution quality. In fact, a larger number of ants ($m \geq 25$) pays off if the algorithm is allowed to run for enough time. This result suggests that increasing the number of ants from an initially low value may lead to better anytime behavior of MMAS.

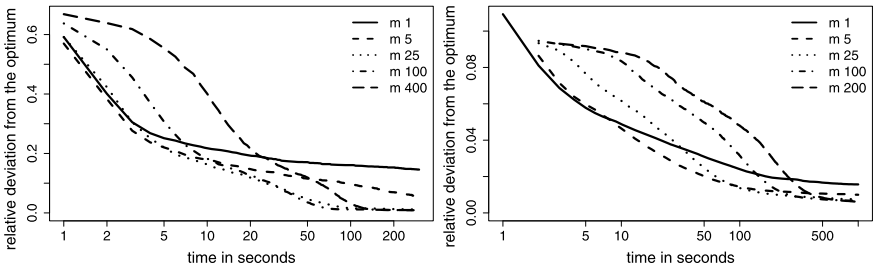

Fig. 8.7: MMAS with various fixed numbers of ants (m); left plot without local search and right plot with local search

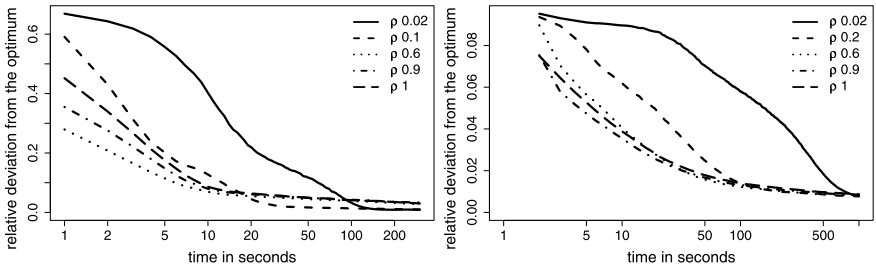

Fig. 8.8: MMAS with various fixed values of ρ; left plot without local search and right plot with local search

ρ, *Figure 8.8:* There is some degree of trade-off between large and small values of ρ. Large values (for example, $\rho = 0.6$) converge faster than the default values ($\rho = 0.02$ without local search, $\rho = 0.2$ with local search). Nevertheless, low values of ρ are able to achieve the same performance, and, given sufficient time, produce the best final results. This effect is most noticeable in the case without local search. Hence, starting with a high evaporation factor and then reducing it over time to its default value appears to be a promising strategy.

q_0, *Figure 8.9:* Finally, we test the use of the pseudorandom proportional rule of ACS (Equation 8.6) in MMAS. Here, we study the effect of different values of q_0 as we previously did for ACS. In this case, a clear trade-off is observed: high values of q_0 perform best for a short runtime, whereas low values of q_0 ($q_0 = 0$ effectively reverts to the standard MMAS) generally result in better final performance.

Summarizing the above experiments, in MMAS a strong trade-off exists for various parameters between the performance of fixed settings for short and long computation times, making the behavior of MMAS very different from that of ACS. In particular, β, m and q_0 seem good candidates for the use of variable settings to achieve good anytime performance. Therefore, in the next section, we examine a few simple ways of varying the parameter settings of MMAS during the run.

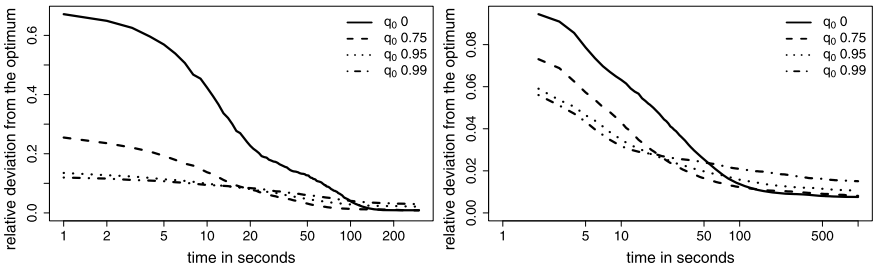

Fig. 8.9: MMAS using the pseudorandom proportional rule with various fixed values of q_0; left plot without local search and right plot with local search

8.6 Prescheduled Parameter Variation for MMAS

Given that MMAS was a clear candidate for the varying of parameters during the computation period, we examine various schedules for changing the parameter settings. In this section, we give exemplary results for prescheduled parameter variation. In particular, we show results concerning the adaptation of the parameters β, m, and q_0. These illustrate the types of improvement in the anytime behavior of MMAS that may be obtained. We do not consider varying the evaporation factor, ρ, since we did not find schedules that significantly improve performance over a fixed, high setting (such as $\rho = 0.6$).

First, we study the variation of β. We tested schedules that decrease the value of β linearly with the iteration counter as well as schedules where a switch from a high value to a low value occurs at a fixed iteration number. The latter type of schedule resulted in better anytime performance and, hence, we focus on these here. The procedure of these schedules is to start with the high value of $\beta = 20$, which was shown to yield good performance at the start of the run, and to later set it directly to a lower value close to the default value. Figure 8.10 shows the results with local search for three scenarios that differ in the number of iterations after which β is changed, from 20 to 3; in particular, we consider 50 (a_β 1), 100 (a_β 2) and 200 (a_β 3) iterations. The schedule a_β 1 obtained the best SQT curve, and delaying the change of β produces worse results. In the case without local search (not shown here), delaying the switch from the high to the low value of β showed some improvement. Nonetheless, for simplicity, we choose strategy a_β 1 for further comparison. Figure 8.11 compares the use of strategy a_β 1 to the use of the default value and a large value ($\beta = 20$) of β without (left) and with (right) local search. In both cases, the prescheduled parameter variation is able to combine the best results of both fixed settings, achieving a better anytime performance.

In the case of the number of ants, m, the strategies studied here start with a single ant and increase the number of ants as the algorithm progresses. Figure 8.12 shows that there is progressive degradation of the quality of the results as the rate at which ants are added increases. The best results are obtained with the lowest rate

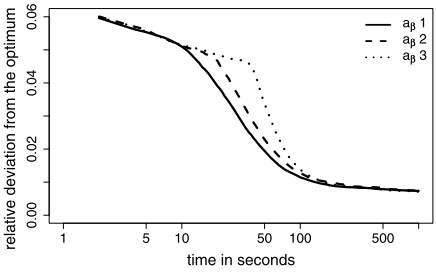

Fig. 8.10: MMAS, scheduled variation of parameter β; the three strategies a_β 1 to a_β 3 start each with β equal to 20 and set β to 3 after 50 (a_β 1), 100 (a_β 2), and 200 (a_β 3) iterations, respectively

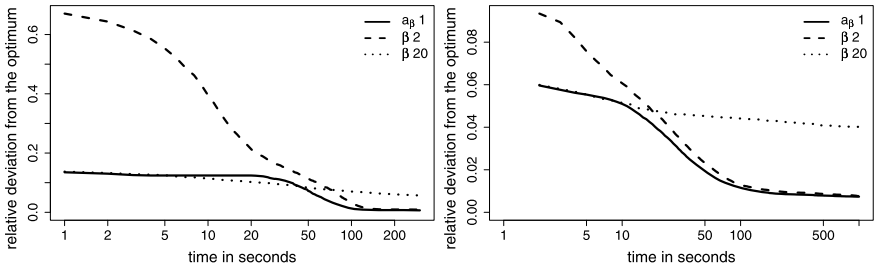

Fig. 8.11: MMAS, comparison of fixed and varying parameter settings for β; left side: case without local search; right side: case with local search. The adaptation strategy used is a_β 1 (see caption of Figure 8.10 for details)

(a_m 1, which adds one ant after every ten iterations) for both cases, with and without local search (only the local search case is shown for conciseness). The comparison between the fixed and prescheduled settings (Figure 8.13) shows a clear benefit with the use of prescheduled variation of m, which matches the good performance obtained for short runtimes by only one ant and for long runtimes by a large number of ants.

For varying q_0 for MMAS, we tested strategies that start with a high value of $q_0 = 0.99$ and decrease it until they reach a setting of q_0 equal to 0. Figure 8.14 shows four strategies that decrease q_0 at different rates, namely, by 0.001 every 15 iterations (a_{q_0} 1), by 0.001 every two iterations (a_{q_0} 2), by 0.005 every iteration (a_{q_0} 3), and by 0.02 every iteration (a_{q_0} 4). Without local search, the strategies that decrease q_0 more slowly result in faster convergence to good solutions (not shown here). However, with local search there is a trade-off between the slowest and the fastest decrease of q_0, with the former being better at the start of the algorithm, and the latter performing best for higher computation times. This suggests that more sophisticated strategies may be able to further enhance the performance of the algorithm. Nevertheless, the comparison of the schedule a_{q_0} 2 with fixed pa-

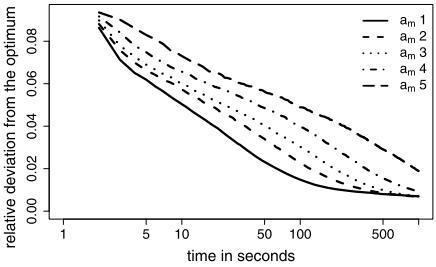

Fig. 8.12: MMAS, scheduled variation of the number of ants (m). All strategies a_m 1 to a_m 5 start with one ant and iteratively increase the number of ants. In particular, a_m 1 adds one ant every ten iterations, a_m 2 adds one ant every second iteration, a_m 3 adds one ant each iteration, a_m 4 adds two ants each iteration, and a_m 5 adds five ants each iteration

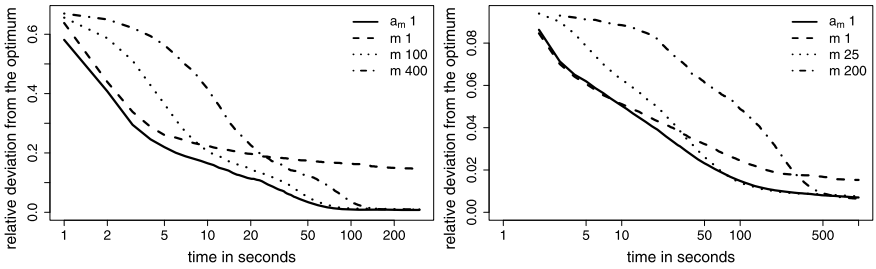

Fig. 8.13: MMAS, comparison of fixed and varying parameter settings for parameter m; left side: case without local search; right side: case with local search. The adaptation strategy used is a_m1 (see caption of Figure 8.12 for details)

rameter settings shows that prescheduled parameter variation is able to match the best results of both fixed parameter settings during the execution time of the algorithm.

A general observation from our study of prescheduled parameter variation is that considerable improvements of the anytime behavior of MMAS are possible without their substantially affecting the final performance achieved by the algorithm. In some additional experiments, we verified that the same conclusion is also true for the parameter α, which weights the influence of the pheromone trails. In fact, for similar simple schedules as proposed previously, we could observe strong improvements in the anytime behavior compared to performance with fixed settings for α. Further studies need to verify whether the same observations on the usefulness of simple prescheduled parameter variations hold for other problems.

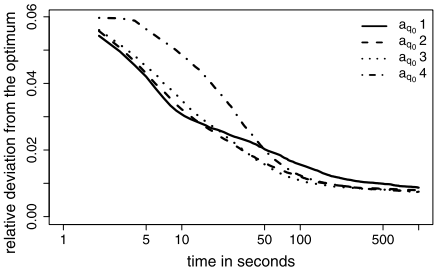

Fig. 8.14: MMAS, scheduled variation of the parameter q_0. All strategies (a$_{q_0}$ 1 to a$_{q_0}$ 4) start at $q_0 = 0.99$ and decrease q_0 to 0. In particular, a$_{q_0}$ 1 decreases q_0 by 0.001 every 15 iterations, a$_{q_0}$ 2 by 0.001 every 2 iterations, a$_{q_0}$ 3 by 0.005 every iteration, and a$_{q_0}$ 4 by 0.02 every iteration

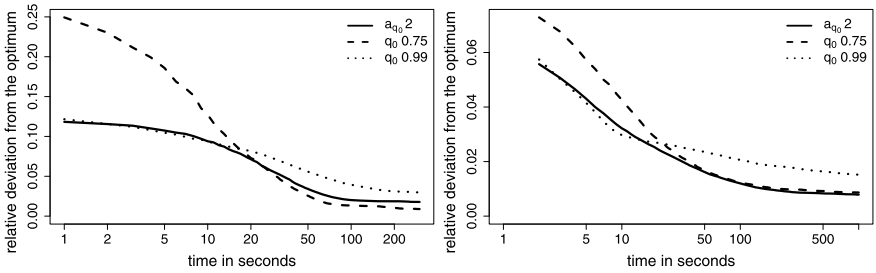

Fig. 8.15: MMAS, comparison of fixed and varying parameter settings for parameter q_0; left side: case without local search; right side: case with local search. The adaptation strategy used is a$_{q_0}$ 2 (see caption of Figure 8.14 for details)

8.7 Conclusions and Future Work

In this chapter, we have given an overview of the literature on parameter adaptation in ACO algorithms. A variety of approaches have been proposed but the overall impression from the research results is that further efforts are required to determine the most suitable strategies for parameter adaptation and their role and importance in ACO algorithms that perform at the state-of-the-art level. Only few of the presented publications have shown clear computational advantages, for example, in the form of better average solution quality reachable in highly effective ACO algorithms.

In the second part of the chapter, we have given an experimental analysis of the impact that specific parameters have on the anytime behavior of ACO algorithms. For the application of ACS and MMAS to the TSP we could determine very different behavior of these algorithms. While the anytime behavior of ACS was rather insensitive to parameter variation, the analysis of the anytime behavior of MMAS has identified clear opportunities for prescheduled parameter variation. We tested a number of fairly straightforward schedules of the values for the parameters β, m,

and q_0 in MMAS. As a result, we could observe that the anytime behavior of MMAS can be greatly improved without significant loss in the final solution quality.

Our computational study can clearly be extended in different directions. An interesting extension is the study of interactions between different parameter settings. Such a study may hint at combined variations of at least two parameters that can further improve the anytime behavior of MMAS or even its final performance. Finally, it is certainly worthwhile studying in more detail the contribution of adaptive strategies that take into account the internal state of the algorithm in order to adapt to different classes of instances. For their study, it is probably preferable to consider problems where the algorithm parameters depend more strongly on specific instance classes than they do in the TSP.

For our main conclusion we can state that parameter adaptation is a relatively large field that is receiving strong attention by the research community. Many techniques have already been proposed in the context of other heuristic methods, but their adoption in ACO algorithms still opens up a number of research opportunities with potentially significant impact.

Acknowledgements This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Mauro Birattari, Thomas Stützle and Marco Dorigo acknowledge support from the Belgian F.R.S.-FNRS, of which they are Research Associates and Research Director, respectively. The authors also acknowledge support from the FRFC project "*Méthodes de recherche hybrides pour la résolution de problèmes complexes*".

References

[1] Amir C., Badr A., Farag I.: A fuzzy logic controller for ant algorithms. Computing and Information Systems 11(2):26–34 (2007)

[2] Anghinolfi D., Boccalatte A., Paolucci M., Vecchiola C.: Performance evaluation of an adaptive ant colony optimization applied to single machine scheduling. In: Li X., et al. (eds.) Simulated Evolution and Learning, 7th International Conference, SEAL 2008, Lecture Notes in Computer Science, vol. 5361, Springer, Heidelberg, Germany, pp. 411–420 (2008)

[3] Battiti R., Brunato M., Mascia F.: Reactive Search and Intelligent Optimization, Operations Research/Computer Science Interfaces, vol. 45. Springer, New York, NY (2008)

[4] Botee H. M., Bonabeau E.: Evolving ant colony optimization. Advances in Complex Systems 1:149–159 (1998)

[5] Cai Z., Huang H., Qin Y., Ma X.: Ant colony optimization based on adaptive volatility rate of pheromone trail. International Journal of Communications, Network and System Sciences 2(8):792–796 (2009)

[6] Chusanapiputt S., Nualhong D., Jantarang S., Phoomvuthisarn S.: Selective self-adaptive approach to ant system for solving unit commitment problem.

In: Cattolico M., et al. (eds.) GECCO 2006, ACM press, New York, NY, pp. 1729–1736 (2006)
[7] Colas S., Monmarché N., Gaucher P., Slimane M.: Artificial ants for the optimization of virtual keyboard arrangement for disabled people. In: Monmarché N., et al. (eds.) Artificial Evolution - 8th International Conference, Evolution Artificielle, EA 2007, Lecture Notes in Computer Science, vol. 4926, Springer, Heidelberg, Germany, pp. 87–99 (2008)
[8] Dorigo M.: Ant colony optimization. Scholarpedia 2(3):1461 (2007)
[9] Dorigo M., Di Caro G.: The Ant Colony Optimization meta-heuristic. In: Corne D., Dorigo M., Glover F. (eds.) New Ideas in Optimization, McGraw Hill, London, UK, pp. 11–32 (1999)
[10] Dorigo M., Gambardella L. M.: Ant Colony System: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1(1):53–66 (1997)
[11] Dorigo M., Stützle T.: Ant Colony Optimization. MIT Press, Cambridge, MA (2004)
[12] Dorigo M., Maniezzo V., Colorni A.: The Ant System: An autocatalytic optimizing process. Tech. Rep. 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Italy (1991)
[13] Dorigo M., Maniezzo V., Colorni A.: Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics - Part B 26(1):29–41 (1996)
[14] Dorigo M., Di Caro G., Gambardella L. M.: Ant algorithms for discrete optimization. Artificial Life 5(2):137–172 (1999)
[15] Dorigo M., Birattari M., Stützle T.: Ant colony optimization: Artificial ants as a computational intelligence technique. IEEE Computational Intelligence Magazine 1(4):28–39 (2006)
[16] Dorigo M., et al. (eds.): Ant Algorithms: Third International Workshop, ANTS 2002, Lecture Notes in Computer Science, vol. 2463. Springer, Heidelberg, Germany (2002)
[17] Eiben A. E., Michalewicz Z., Schoenauer M., Smith J. E.: Parameter control in evolutionary algorithms. In: [31], pp. 19–46 (2007)
[18] Favaretto D., Moretti E., Pellegrini P.: On the explorative behavior of MAX–MIN Ant System. In: Stützle T., Birattari M., Hoos H. H. (eds.) Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics. SLS 2009, Lecture Notes in Computer Science, vol. 5752, Springer, Heidelberg, Germany, pp. 115–119 (2009)
[19] Förster M., Bickel B., Hardung B., Kókai G.: Self-adaptive ant colony optimisation applied to function allocation in vehicle networks. In: Thierens D., et al. (eds.) GECCO'07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, pp. 1991–1998 (2007)
[20] Gaertner D., Clark K.: On optimal parameters for ant colony optimization algorithms. In: Arabnia H. R., Joshua R. (eds.) Proceedings of the 2005 International Conference on Artificial Intelligence, ICAI 2005, CSREA Press, pp. 83–89 (2005)

[21] Gambardella L. M., Dorigo M.: Ant-Q: A reinforcement learning approach to the traveling salesman problem. In: Prieditis A., Russell S. (eds.) Proceedings of the Twelfth International Conference on Machine Learning (ML-95), Morgan Kaufmann Publishers, Palo Alto, CA, pp. 252–260 (1995)
[22] Garro B. A., Sossa H., Vazquez R. A.: Evolving ant colony system for optimizing path planning in mobile robots. In: Electronics, Robotics and Automotive Mechanics Conference, IEEE Computer Society, Los Alamitos, CA, pp. 444–449 (2007)
[23] Hao Z., Cai R., Huang H.: An adaptive parameter control strategy for ACO. In: Proceedings of the International Conference on Machine Learning and Cybernetics, IEEE Press, pp. 203–206 (2006)
[24] Hao Z., Huang H., Qin Y., Cai R.: An ACO algorithm with adaptive volatility rate of pheromone trail. In: Shi Y., van Albada G. D., Dongarra J., Sloot P. M. A. (eds.) Computational Science – ICCS 2007, 7th International Conference, Proceedings, Part IV, Lecture Notes in Computer Science, vol. 4490, Springer, Heidelberg, Germany, pp. 1167–1170 (2007)
[25] Hoos H. H., Stützle T.: Stochastic Local Search–Foundations and Applications. Morgan Kaufmann Publishers, San Francisco, CA (2005)
[26] Khichane M., Albert P., Solnon C.: An ACO-based reactive framework for ant colony optimization: First experiments on constraint satisfaction problems. In: Stützle T. (ed.) Learning and Intelligent Optimization, Third International Conference, LION 3, Lecture Notes in Computer Science, vol. 5851, Springer, Heidelberg, Germany, pp. 119–133 (2009)
[27] Kovářík O., Skrbek M.: Ant colony optimization with castes. In: Kurkova-Pohlova V., Koutnik J. (eds.) ICANN'08: Proceedings of the 18th International Conference on Artificial Neural Networks, Part I, Lecture Notes in Computer Science, vol. 5163, Springer, Heidelberg, Germany, pp. 435–442 (2008)
[28] Li Y., Li W.: Adaptive ant colony optimization algorithm based on information entropy: Foundation and application. Fundamenta Informaticae 77(3):229–242 (2007)
[29] Li Z., Wang Y., Yu J., Zhang Y., Li X.: A novel cloud-based fuzzy self-adaptive ant colony system. In: ICNC'08: Proceedings of the 2008 Fourth International Conference on Natural Computation, IEEE Computer Society, Washington, DC, vol. 7, pp. 460–465 (2008)
[30] Ling W., Luo H.: An adaptive parameter control strategy for ant colony optimization. In: CIS'07: Proceedings of the 2007 International Conference on Computational Intelligence and Security, IEEE Computer Society, Washington, DC, pp. 142–146 (2007)
[31] Lobo F., Lima C. F., Michalewicz Z. (eds.): Parameter Setting in Evolutionary Algorithms. Springer, Berlin, Germany (2007)
[32] Martens D., Backer M. D., Haesen R., Vanthienen J., Snoeck M., Baesens B.: Classification with ant colony optimization. IEEE Transactions on Evolutionary Computation 11(5):651–665 (2007)
[33] Melo L., Pereira F., Costa E.: MC-ANT: A multi-colony ant algorithm. In: Artificial Evolution - 9th International Conference, Evolution Artificielle, EA

2009, Lecture Notes in Computer Science, vol. 5975, Springer, Heidelberg, Germany, pp. 25–36 (2009)
[34] Merkle D., Middendorf M.: Prospects for dynamic algorithm control: Lessons from the phase structure of ant scheduling algorithms. In: Heckendorn R. B. (ed.) Proceedings of the 2000 Genetic and Evolutionary Computation Conference - Workshop Program. Workshop "The Next Ten Years of Scheduling Research", Morgan Kaufmann Publishers, San Francisco, CA, pp. 121–126 (2001)
[35] Merkle D., Middendorf M., Schmeck H.: Ant colony optimization for resource-constrained project scheduling. IEEE Transactions on Evolutionary Computation 6(4):333–346 (2002)
[36] Meyer B.: Convergence control in ACO. In: Genetic and Evolutionary Computation Conference (GECCO), Seattle, WA, late-breaking paper available on CD (2004)
[37] Pellegrini P., Favaretto D., Moretti E.: Exploration in stochastic algorithms: An application on MAX–MIN Ant System. In: Nature Inspired Cooperative Strategies for Optimization (NICSO 2008), Studies in Computational Intelligence, vol. 236, Springer, Berlin, Germany, pp. 1–13 (2009)
[38] Pilat M. L., White T.: Using genetic algorithms to optimize ACS-TSP. In: [16], pp. 282–287 (2002)
[39] Randall M.: Near parameter free ant colony optimisation. In: Dorigo M., et al. (eds.) Ant Colony Optimization and Swarm Intelligence: 4th International Workshop, ANTS 2004, Lecture Notes in Computer Science, vol. 3172, Springer, Heidelberg, Germany, pp. 374–381 (2004)
[40] Randall M., Montgomery J.: Candidate set strategies for ant colony optimisation. In: [16], pp. 243–249 (2002)
[41] Stützle T.: ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem. URL http://www.aco-metaheuristic.org/aco-code/ (2002)
[42] Stützle T., Hoos H. H.: MAX–MIN Ant System. Future Generation Computer Systems 16(8):889–914 (2000)
[43] White T., Pagurek B., Oppacher F. Connection management using adaptive mobile agents. In: Arabnia H. R. (ed.) Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98), CSREA Press, pp. 802–809 (1998)
[44] Zilberstein S.: Using anytime algorithms in intelligent systems. AI Magazine 17(3):73–83 (1996)
[45] Zlochin M., Birattari M., Meuleau N., Dorigo M.: Model-based search for combinatorial optimization: A critical survey. Annals of Operations Research 131(1–4):373–395 (2004)