 ISBN: 978-989-95618-4-7

Multi-caste Ant Colony Optimization Algorithms

Leonor Melo[1,2], Francisco Pereira[1,2] and Ernesto Costa[2]

[1] Instituto Superior de Engenharia de Coimbra, 3030-199 Coimbra, Portugal
[2] Centro de Informática e Sistemas da Universidade de Coimbra, 3030-790 Coimbra, Portugal
leonor@isec.pt, xico@dei.uc.pt, ernesto@dei.uc.pt

Abstract. In this paper we present a multi-caste ant colony optimization approach, where each caste has its own set of parameters. Two variants are proposed: in the first, the composition of the castes remains fixed throughout the optimization, whilst the other allows ants to move from one caste to another. Results obtained in several traveling salesperson problem instances reveal that the adoption of a multi-caste framework increases the robustness of ACO algorithms. In concrete, we show that the existence of different castes removes the need to carefully define q_0, an essential parameter for the success of Ant Colony System.

Keywords: Ant Colony Optimization, Parameters Adaptation, Multiple Castes

1 Introduction

Ant colony optimization (ACO) algorithms are powerful metaheuristics loosely inspired in the social behavior of ants. When foraging, real ants deposit pheromone on the ground, thereby signaling relevant information to other members of the colony. This indirect communication mechanism allows ants to collectively solve difficult tasks. ACO algorithms are computational models of this behavior. They are iterative methods based on a set of artificial agents that cooperatively solve difficult optimization problems. Communication is implemented as pheromone signals that artificial ants place on promising solutions components.

Actually, there are many different ACO variants with differences, e.g., in the way pheromone levels are updated. Selecting the best ACO configuration to apply in a specific optimization situation is not trivial. It requires a deep understanding of the properties of different ACO variants and a careful selection of the settings to adopt. In this paper we focus on the task of determining the best set of parameters. It is well-known that the behavior of ACO methods strongly depends on the selected settings. Moreover, even a perfect set of parameters for the early stages of optimization, might turn out to be a poor choice as the run progresses. With this is mind, we propose a multi-caste ACO framework that removes the need to perform a careful parameter specification. The colony is

divided in several castes, each one with a specific set of parameters. Two multi-caste variants are considered: in the first, the composition of the castes remains fixed, whereas in the second, ants are allowed to jump from one caste to another in search of a better set of parameters.

The multi-caste framework is used to gain insight about the advantages of relying on adjustable settings for the Ant colony systems (ACS), one of the main ACO variants. We focus our attention on q_0, a parameter that strongly influences the behavior of the ACS [18]. Results obtained with the traveling salesperson problem (TSP) reveal that the multi-caste approach increases the robustness of the ACS variant. The standard mono-caste version is highly dependent on the specific q_0 value selected, and the best results on distinct TSP instances are obtained with different settings. On the contrary, the multi-caste versions of ACS exhibit a similar behavior irrespectively of the specific q_0 values ascribed. This outcome shows that the increased diversity introduced by castes helps ACS to avoid low quality traps in the search space.

The paper is structured as follows: in section 2 we briefly describe ACO algorithms. Section 3 comprises a presentation of the multi-caste ACS variants. In section 4 we present and analyze the main optimization results. Finally, section 5 gathers the main conclusions and highlight directions for future work.

2 Ant Colony Optimization

ACO is one of the most successful branches of swarm intelligence. ACO algorithms were originally proposed by Marco Dorigo and, as its name suggests, take inspiration from pheromone-based interactions occurring in ant societies [3]. Real ants deposit pheromone on the ground while foraging. The other ants from the colony tend to follow the path where the pheromone concentration is higher, collectively leading to the appearance of a promising trail [11].

ACO algorithms mimic this behavior when solving an optimization problem. A set of artificial ants iteratively build solutions for a given situation. Relevant information is shared, thereby increasing the likelihood of discovering promising solutions. In concrete, an artificial ant is a probabilistic procedure that constructs a solution biased by heuristic information and pheromone values. The heuristic knowledge is usually modeled as greedy information, specific from the problem being solved. The pheromone values represent dynamic feedback information, reflecting the colony search experience and implementing a mechanism for indirect communication. Pheromone values change over time, guiding the search procedure towards promising solutions.

2.1 Ant Colony Optimization for the TSP

The TSP is a famous NP-hard combinatorial optimization problem. Given a set of cities and all pairwise distances between them, the goal is to discover the shortest tour that visits every city exactly once. This was the first problem

addressed by ACO algorithms, both because it is a difficult optimization situation and it can be modeled in a suitable way for the exploration performed by artificial ants. A specific TSP instance is represented by a fully connected undirected graph $G = (V, E)$ where V is the set of cities and E is the set of roads connecting pairs of cities. Each edge from E has a distance e_{ij} associated. The objective is to find the Hamiltonian cycle of minimal total cost. To apply an ACO algorithm to a problem, one must first define the solution components. A connected graph is then created by associating each component with a vertex and creating edges to link vertices. The TSP representation previously defined immediately establishes the graph where the ants will operate. The pheromone value, τ_{ij}, represents the desirability of a certain edge. The higher the value, the more attractive that edge is for the ants. Associated with each edge there is also a static heuristic value, $\eta_{ij} = \frac{1}{e_{ij}}$, representing the attractiveness of the edge from a greedy point of view.

Ants start building a solution in a random vertex and iteratively add components by following a specific edge. At each decision point (the current vertex), an ant makes a probabilistic choice of the path to take (only edges that do not violate feasibility constraints are considered). The choice is biased by the pheromone level and heuristic knowledge of each possible path. After completing a solution, ants provide feedback by depositing pheromone on the edges crossed. The goal of pheromone update is to reinforce the desirability of paths appearing in promising solutions. To avoid stagnation, pheromone levels are periodically decreased by a certain factor.

2.2 Ant Colony System (ACS)

Ant System (AS) was the first ACO algorithm proposed [5], [6]. In this paper we focus on Ant Colony System (ACS), an AS variant that was proposed with the aim to improve efficiency when applied to the TSP [4]. ACS differs form AS in 3 aspects:

State transition rules When deciding which edges should be part of the solution the ants use a set of rules. In ACS, these rules allow for choosing between the exploration of new solutions or the exploitation of heuristic and acquired knowledge. An ant in node i decides to move to node j according to the rule in equation (1)

$$j = \begin{cases} argmax_{il \in N(s^p)} \left\{ \tau_{il} \cdot \eta_{il}^{\beta} \right\} & \text{if } q < q_0 \quad \text{(exploitation)} \\ J & \text{otherwise (biased exploration)} \end{cases} \tag{1}$$

where q is a uniformly distributed variable over $[0, 1]$, $q_0 \in [0, 1]$ is a parameter, $N(s^p)$ is the set of cities not visited yet, τ_{il} is the pheromone value associated with that move, η_{il} is the corresponding heuristic information, β is a parameter used to control the relative influence of the heuristic information, and J is a random

variable selected according to the probability distribution given by equation (2).

$$p_{ij} = \begin{cases} \frac{\tau_{ij} \cdot \eta_{ij}^{\beta}}{\sum_{il \in N(s^p)} \tau_{il} \cdot \eta_{il}^{\beta}} & \text{if } c_{ij} \in N(s^p) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where p_{ij} is the probability of an ant in node i to move to node j. The rules displayed in (1) and (2) favor the transition to nodes that are near and have a connection with a high pheromone level. The parameter q_0 is essential in ACS, as it balances the relative importance given to exploration versus exploitation. Whenever an ant has to make a decision about which path to follow, a variable q is sampled: if $q < q_0$ the path with the highest $\tau_{il} \cdot \eta_{il}^{\beta}$ value is chosen (exploitation); otherwise, an edge is probabilistically selected according to (2)(biased exploration).

Global pheromone updating rule In the end of an iteration, only the best ant (since the beginning of the run) is allowed to update the trail. The logical reason to this restriction is to bias the search towards area surrounding the best solution found so far. The trail is updated according to equation (3)

$$\tau_{ij} = \begin{cases} (1-\rho) \cdot \tau_{ij} + \rho / L_{gb} & \text{if } c_{ij} \text{ belongs to the solution} \\ \tau_{ij} & \text{otherwise} \end{cases} \quad (3)$$

where $0 < \rho < 1$ is the pheromone decay parameter and L_{gb} is the length of the best solution found so far.

Local pheromone updating rule Each time an edge is crossed by an ant, its pheromone value is slightly decreased, thereby discouraging remaining ants to follow the exact same trail in that iteration. The objective of the local pheromone updating is to prevent excessive convergence and to promote the exploration of alternative solutions inside an iteration. After passing through an edge, an ant updates the pheromone level using the formula (4)

$$\tau_{ij} \leftarrow (1-\xi) \cdot \tau_{ij} + \xi \cdot \tau_0 \quad (4)$$

where $0 < \xi < 1$ is the pheromone decay parameter and τ_0 is the initial pheromone level.

The application of ACS to the optimization of a given problem, requires the definition of the following parameters:

- m - Number of ants in the colony;
- β - relevance given to the heuristic knowledge in the selection of an edge;
- ρ - evaporation rate: if ρ is low the pheromone values will persist longer.
- q_0 - probability of selecting the next city greedily;
- ξ - pheromone decay coefficient: used to make the trails already used in the present iteration less attractive to the other ants;
- τ_0 - initial pheromone value for all the sections of the trail: usually set to a very small constant value.

3 Multi-caste Ant Colony Optimization

Though being a successful metaheuristic, the behavior of ACO algorithms strongly depends on the values of some parameters [7], [8], [18]. However, discovering the ideal settings for a specific situation is a non-trivial task and it requires significant efforts. The most straightforward approach is to perform an off-line parameter setting, i.e., to estimate what might be the best set of values before applying the algorithm in the real optimization task. This alternative is usually accomplished by a trial-and-error procedure, and is a time-consuming, human-intensive and error prone task [1], [20]. Moreover, different problems (and even different instances of the same problem) usually require different settings, and therefore previous analysis might not be helpful in new situations.

Adaptive parameter tuning is an active area of research (see, e.g., [18], [20] and references therein). In simple terms, this approach consists in self-adjusting the parameter values while the algorithm is running. The advantages are clear: self-adaptation removes the need to engage in cumbersome estimations about what might be the ideal setting for a specific situation and it contributes to the appearance of more robust ACO algorithms, i.e., algorithms that self-tailor to the problem being addressed. Obviously, there is also a limitation since the algorithm must adjust its setting, while simultaneously searching for promising solutions.

In this paper we present a set of experiments that help to gain insight about the advantages of relying on an adaptive setting for the ACS. We focus our attention on the q_0 parameter, as this is the most distinctive feature of this ACO variant and has a very high influence in the behavior of the algorithm [18]. In the next sections we briefly highlight the limitations of fixed settings and introduce two simple ACS variants that allow for an on-the-fly adaptation of q_0.

3.1 Limitations of ACS with fixed settings

Varying the parameter values as the search progresses might lead to an overall performance enhancement [12]. The work described in [20] studies the evolution of the solution quality as the computations run, for different parameter settings. Even though the analysis is mainly focused on the MAX-MIN variant, the general conclusion is that the best parameter values depend on the specific point of the search process where the algorithm is at. We did a set of similar experiments for ACS, focusing on the impact of adopting different values for $q_0 = \{0.75, 0.9, 0.95, 0.99\}$. The anytime-behavior [22] of the ACS was recorded for several instances of the TSP. An illustrative example of the type of results obtained may be observed in Fig. 1. The chart displays the evolution of the mean-best-fitness (MBF) for each value of q_0, in a TSP instance with 417 cities. Results are averages of 30 runs, obtained with the same parameter settings, and the y axis shows the deviation of the MBF from the optimum. ACS didn't use local search, in order to better understand the real influence of parameter q_0 in the behavior of the algorithm.

 ISBN: 978-989-95618-4-7

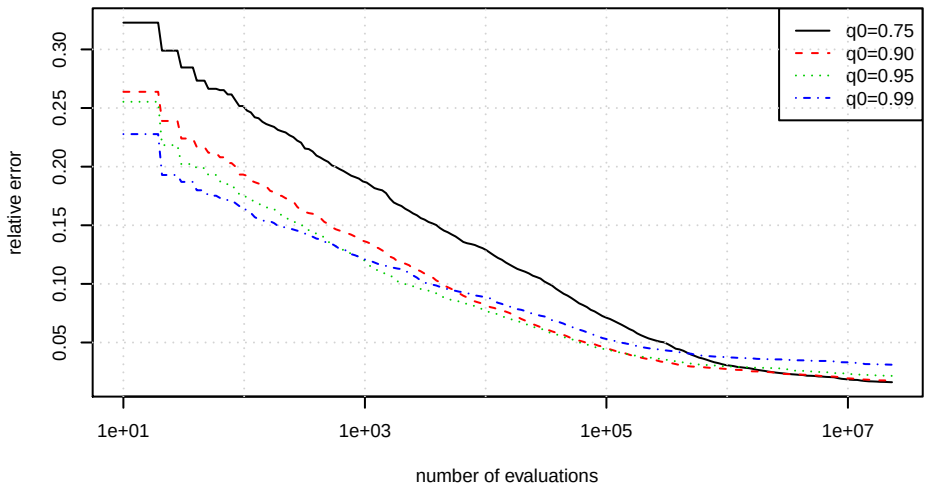

Fig. 1. Example of the anytime behavior of ACS for an instance of size 417

The results clearly show the impact of q_0 in the performance of ACS. In the beginning of the run, higher values of q_0 lead to a fast discovery of promising solutions. However, excessive greediness leads to premature convergence. Lower values of q_0 allow the ACS to maintain the ability to improve solutions and, by the time the optimization reaches 10^7 evaluations, $q_0 = 0.75$ is already the configuration. Experiments performed on instances of different size revealed the same trend. We then argue that it might be advantageous to let ACS switch its q_0 during the optimization, as this will enable the algorithm to adjust to different optimization stages. The next ACS variants establish how this adjustment can be done.

3.2 Multi-caste ACS

In the multi-caste version of the ACS, a colony has more than one caste of ants. Ants belonging to different castes have different parameter values. The idea behind our proposal is to allow for the algorithm to use the best ant for any given moment. We propose two variants: const-multi-caste and jump-multi-caste. The alterations needed to the conventional ACS are minimal.

Const-multi-caste The colony is divided in castes, all with the same number of ants. The distribution remains fixed throughout the optimization. Each caste is characterized by a specific q_0 value. Ants inherit the setting from the caste to which they belong and, when applying the state transition rules, rely on their specific q_0 value.

Jump-multi-caste The initial distribution is similar to that of const-multi-caste. However, at the end of each iteration, two ants are selected at random. If

the ants belong to different castes and both castes have more than 20% of the total number of ants, the quality of their solutions is compared. The ant with the worse solution jumps to the caste of the winning ant. The idea behind this variant is to provide a simple method to dynamically adjust the size of the castes, favoring those that in the current search status encode the most promising q_0 value.

3.3 Related Work

Several approaches were proposed in the past few years to address the adjustment of parameters values throughout the execution of the algorithm. Whereas in some proposals, parameter values change according to some predefined schedule or are function of the time or number of iterations ([13], [20]), in others, the adaptation depends on the behavior of the algorithm, e.g., on the distance between the present solution and the optimum ([16], [2], [9]). Still, other approaches expand the search space by adding the dimensions that represent the parameters ([21], [15]) or rely on an external search procedure to optimize the parameters values during the execution of the algorithm([10], [12], [14]). For a recent and detailed overview of the different methods, please refer to [20].

4 Experiments

Several experiments were performed to compare the results obtained by const-multi-caste and jump-multi-caste against the conventional ACS. The goal is to gain insight about the possible advantages of relying on ACS variants that allow for the simultaneously existence of different q_0 values and that adjust the relative weight of each one of the specific settings considered. We used the publicly available ACOTSP software [19], both to get the values of the standard ACS variant and as the base for our own implementations. Unless otherwise noted, the default values used for the experiments are: $m = 10$, $\beta = 2$, $\rho = 0.1$, $\xi = 0.1$, $\tau_0 = 1/(n \cdot L_{nn})$ (where L_{nn} is the length of the tour using the nearest neighbor heuristic [7], [19]), $q_0 = \{0.75, 0.9, 0.95, 0.99\}$. Local search was never used and each experiment was repeated 30 times.

Several symmetric TSP instances of different size were selected from the TSPLIB 95 [17]. In concrete, we performed experiments with instances with 99, 189, 417, 783, 1577, 3038 and 5934 cities, for which the optimal solution is known. To better understand the effect of the multi-caste approach we divided the configurations in 3 groups as depicted in Table 1. The mono_caste group uses the conventional ACS algorithm, the multi_caste group uses the const-multi-caste variant and the jump_caste group the jump-multi-caste. All the configurations set the total number of ants, m, to 10, except the cquads and fquads configurations that have $m = 20$ (so that all castes begin with 5 ants). The values for q_0 are the ones noted in the last column of the table. All tests were performed on an ATHLON 64 X2 3800+ 2.0 GHz computer and allowed to run for 5000 seconds for most instances and 20000 seconds for rl5934.

Table 1. Configurations used in the experiments

group	configuration	n. of castes	n. of ants per caste	q_0 of the castes
mono_caste	c75	1	10	0.75
	c90	1	10	0.90
	c95	1	10	0.95
	c99	1	10	0.99
multi_caste	c75_90	2	5	0.75 and 0.90
	c75_95	2	5	0.75 and 0.95
	c75_99	2	5	0.75 and 0.99
	c90_95	2	5	0.90 and 0.95
	c90_99	2	5	0.90 and 0.99
	c95_99	2	5	0.95 and 0.99
	cquads	4	5	0.75, 0.90, 0.95 and 0.99
jump_caste	j75_90	2	5	0.75 and 0.90
	j75_95	2	5	0.75 and 0.95
	j75_99	2	5	0.75 and 0.99
	j90_95	2	5	0.90 and 0.95
	j90_99	2	5	0.90 and 0.99
	j95_99	2	5	0.95 and 0.99
	jquads	4	5	0.75, 0.90, 0.95 and 0.99

4.1 Results

Table 2 contains an overview of the optimization results. Columns 3 to 9 identify the ACS configuration that obtained the best and worst MBF in the 7 TSP instances used in the experiments. In each column, symbol **V** marks the configuration with the absolute best average performance, whereas symbol ✓marks the variants that obtained results with a distribution identical to the overall best performing configuration (significance level of 0.05). Likewise, symbol **X** marks the worst configuration and X highlights variants who have an identical distribution to **X**.

The column corresponding to the rat99 instance is not very informative. This is a small instance and nearly all ACS variants obtained results that are statistically identical to the best performing configuration. Differences are so small, that several variants obtained results that are statistically identical to the best and worst configurations. Focusing on the other 6 instances, some interesting patterns arise. In most cases, the absolute best and worst performing configurations are from the mono_caste group. This reveals both a strength and a weakness of standard ACS. If one is able to find ideal parameter settings, then the algorithm excels in the optimization. However, a poor selection for the value of a single parameter severely compromises its effectiveness. Information from Table 2 also confirms that offline tuning is a difficult task: three different values of q_0 obtained the best results with mono_caste in different TSP instances. As for the const_multi_caste and jump_caste groups, they show signs of increased robustness. They never obtain the worst result and, with just a few exceptions,

they do not achieve results identical to the worst. Also, they rarely exhibit the absolute best performance, but they frequently obtain results that are similar to those achieved by the best performing configuration. This is an interesting result. A straightforward modification in the structure of ACS, allowing for several q_0 to coexist during the optimization, visibly increases the robustness of the algorithm, preventing it from getting trapped in low quality regions of the search space.

Table 2. Overview of the optimization results in the 7 selected TSP instances. Symbol **V** marks the best configuration and ✓highlights the variants who's distribuition isn't have an identical distribution to **V**. Symbol **X** marks the worst configuration and X highlights variants who have an identical distribution to **X**.

group	configuration	rat99	d198	fl417	rat783	fl1577	pcb3038	rl5934
	c75	**X**	✓	**V**	**X**	✓	**X**	**X**
mono_caste	c90	✓X	**V**	✓	✓	**V**		
	c95	✓X			✓		**V**	**V**
	c99	✓X	**X**	**X**		**X**		
	c75_90	✓X		✓		✓		
	c75_95	✓X	✓		✓	✓		
	c75_99	✓X				✓		
multi_caste	c90_95	✓X		✓	✓	✓X	✓	
	c90_99	✓X			✓	✓X		
	c95_99	✓			✓			✓
	cquads	**V**	✓			X		
	j75_90	✓X	✓	✓				
	j75_95	✓X	✓	✓	**V**			
	j75_99	✓		X		X		
jump_caste	j90_95	✓X	✓	✓	✓	✓	✓	✓
	j90_99	✓X	✓			X		✓
	j95_99	✓	✓	X		X		✓
	jquads	✓			✓	X		

Results do not show a clear advantage of quads configurations (cquads and jquads) over those that only consider two q_0 values. This is true both for multi_caste and jump_caste groups. With the current results, it is impossible to determine if this is a consequence of the basic framework proposed for multi-caste variants or is an indicator that just two settings are enough to allow for an increased robustness in search. It is important to notice that the quads configurations requires 20 ants (instead of the 10 used by the other configurations). Usually, the performance of ACS tends to deteriorate as the number of ants increases and this might compromise the performance of the quads. Understanding the impact of the number of castes in the performance of ACS is a topic that we will address in future research. Experimental results show that the jump-multi-caste approach does not provide any clear advantage over const-multi-caste, thereby suggesting

 ISBN: 978-989-95618-4-7

that on-the-fly adaptation of the number of ants belonging to each caste is not useful. This might be a consequence of the basic adjustment procedure proposed for jump-multi-caste and we believe that a different adaptive strategy might help to further enhance the robustness of this approach.

A detailed distribution of results can be consulted in the box-plot charts from Figures 2 to 8. This detailed view confirms the general trends derived from the analysis of data in table 2. As a rule, there is always a mono_caste group that is clearly worse that the others in the same group, and many times worse than any other in any other group. In the groups multi_caste and jump_caste, the difference between the best and worst performing configurations is not so noticeable. The outcome of the fl1577 instance is somehow atypical, as the differences between the best and worse performing configurations are not as marked. This is probably a consequence of the specific properties of this particular instance.

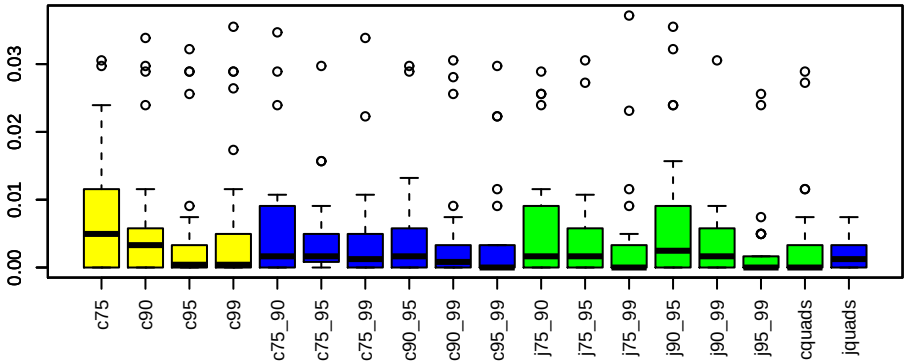

Fig. 2. Results for the rat99 instance

Figure 9 displays the anytime behavior of the const-multi-caste variant for the same instance that was used to study the same behavior of standard ACS (see Fig. 1). A comparison of both figures reveals that multi-caste ACS tends to be more stable and with smaller differences in performance than those exhibited by standard ACS with different q_0 values. More important, over time the different configurations tend to converge to the approximately the same good results, therefore making parameter selection less relevant to the success of the optimization. Even though we display results from just one instance, the same trend is visible in other cases.

We complete our analysis by providing evidence that the multi_caste variants are indeed using different q_0 values. Figure 10 displays a chart with the value of q_0 encoded in the best so far ant over time. Results are averages of 30 runs and were obtained in the instance with 783 cities, although the same trend is visible for other instances. It can be seen that the different castes contribute to discover new best solutions, as the q_0 keeps oscillating between extremes. In

ISBN: 978-989-95618-4-7

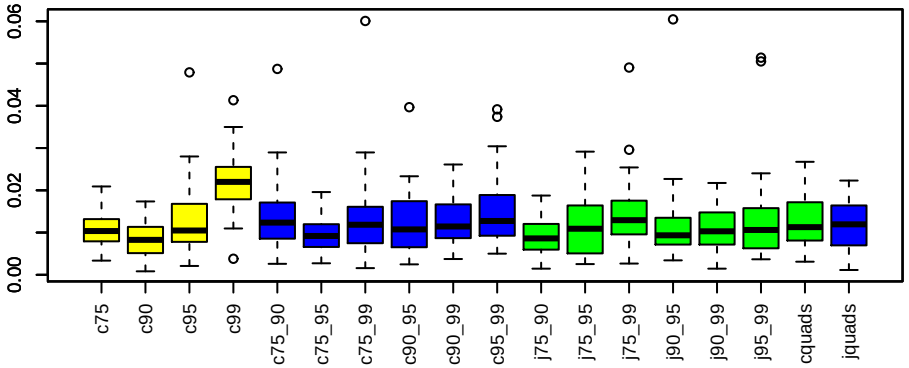

Fig. 3. Results for the d198 instance

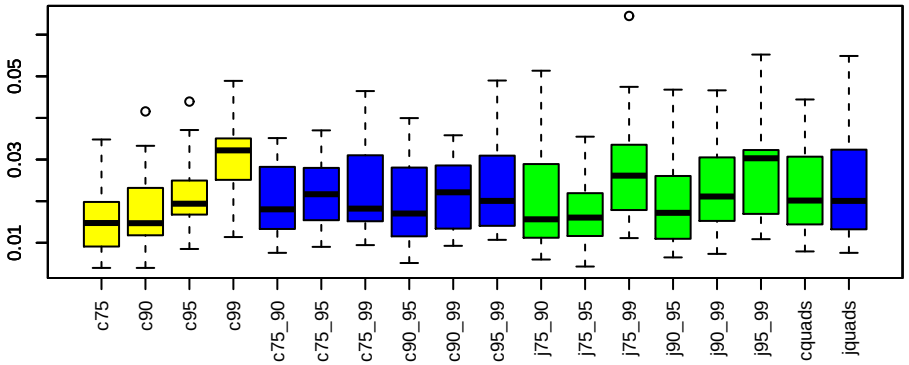

Fig. 4. Results for the fl417 instance

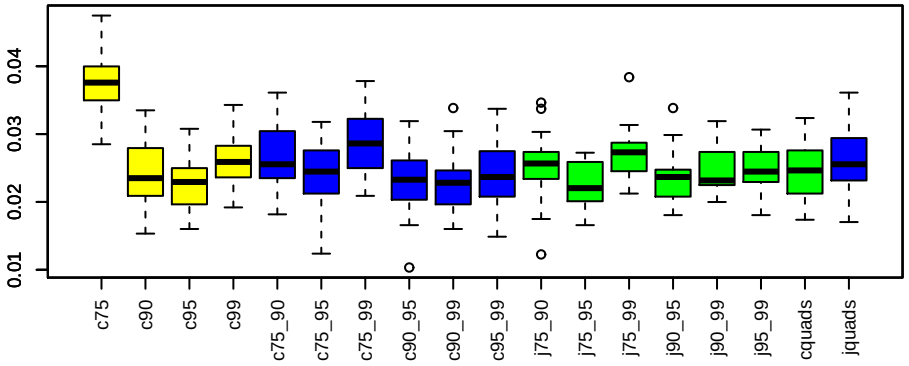

Fig. 5. Results for the rat783 instance

ISBN: 978-989-95618-4-7

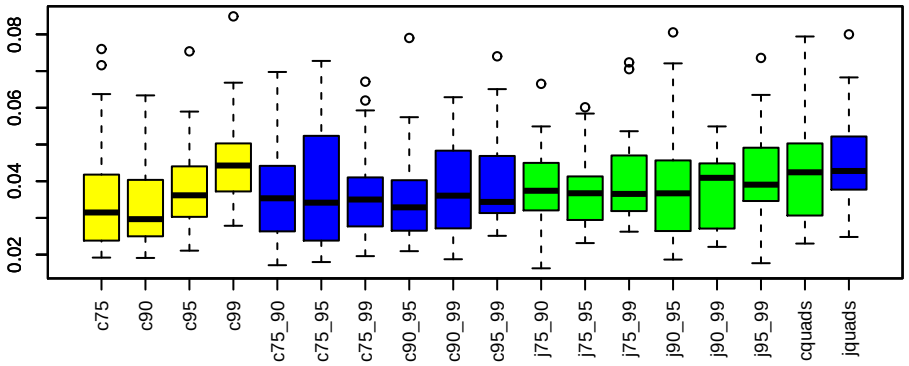

Fig. 6. Results for the fl1577 instance

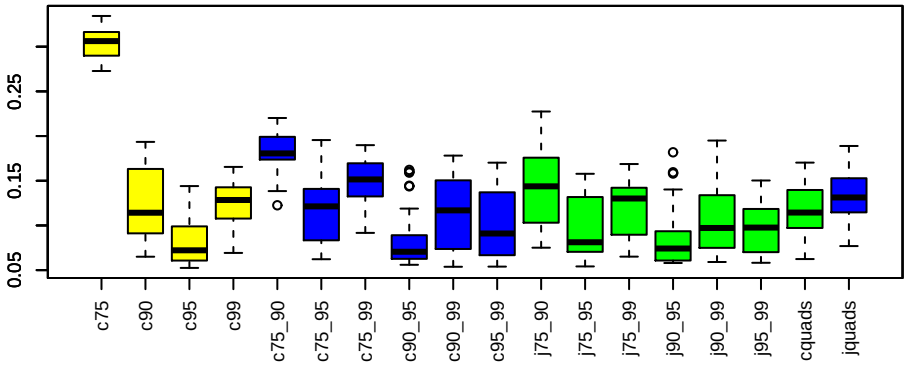

Fig. 7. Results for the pcb3038 instance

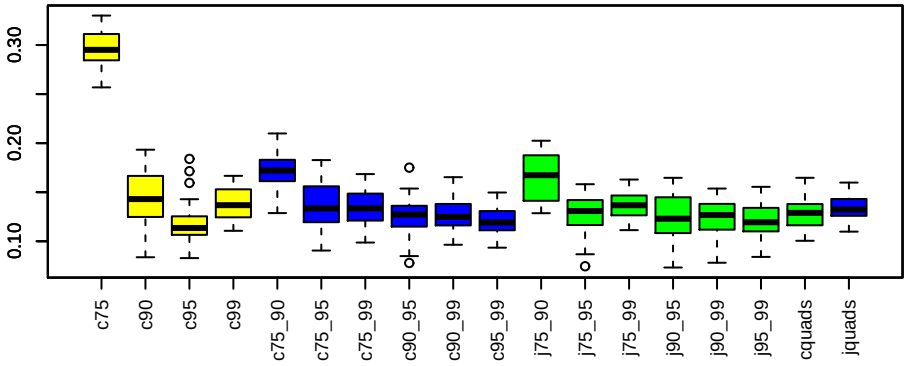

Fig. 8. Results for the rl5934 instance

 ISBN: 978-989-95618-4-7

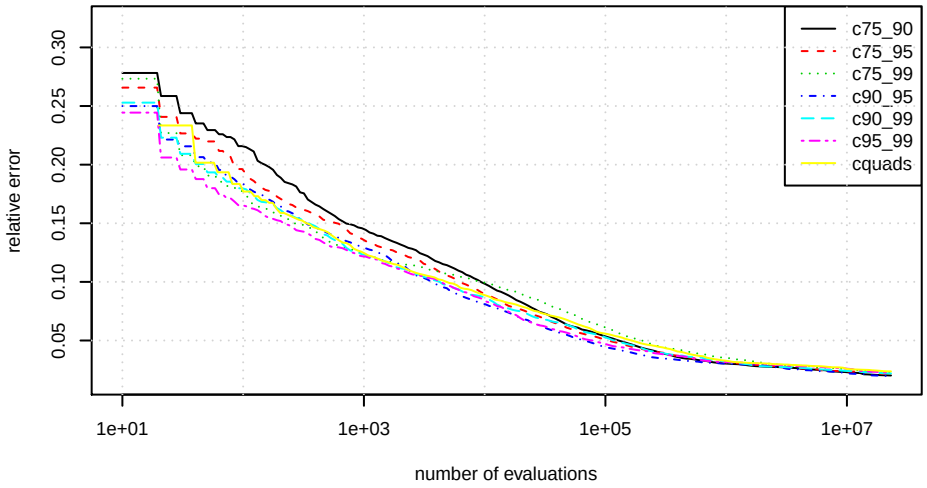

Fig. 9. Example of the anytime behavior of multi_caste for an instance of size 417

most instances there is a slight trend for the average value of q_0 to decrease over time. This is in accordance with the analysis of the anytime behavior of standard ACS, which revealed an advantage of using lower values of q_0 as the search progresses.

5 Conclusions

Parameter settings play an important role in the performance of ACO algorithms and, in recent years, there has been a growing interest in the development of adaptive approaches that adjust settings during the optimization. We presented and analyzed two multi-caste variants of ACS that allow for different values of q_0 to be used during a single run of the algorithm. The first variant, const-multi-caste, contains several castes with a fixed number of ants, each group with its own q_0 value. Jump-multi-caste also relies on different castes, but allows ants to migrate from one group to the other in search of a more effective setting for a specific period of the optimization. Results obtained with several TSP instances reveal that multi-caste configurations are more robust than standard ACS. They are particularly effective in avoiding the poor performance that results from a suboptimal selection of parameters, since they are able to discover good solutions irrespectively of the exact configuration adopted. This is a relevant result, as it simplifies the task of specifying the parameters for an ACO algorithm and increases the likelihood of discovering promising solutions.

Experimental results are not conclusive about the advantage of relying on jump-multi-caste over the const-multi-caste variant. A different adaptive strategy might be required for jump-multi-caste and this is a topic that we will address in the near future. Also, we aim to expand this framework to other parameters, as this might help to further enhance the robustness of the adaptive algorithm.

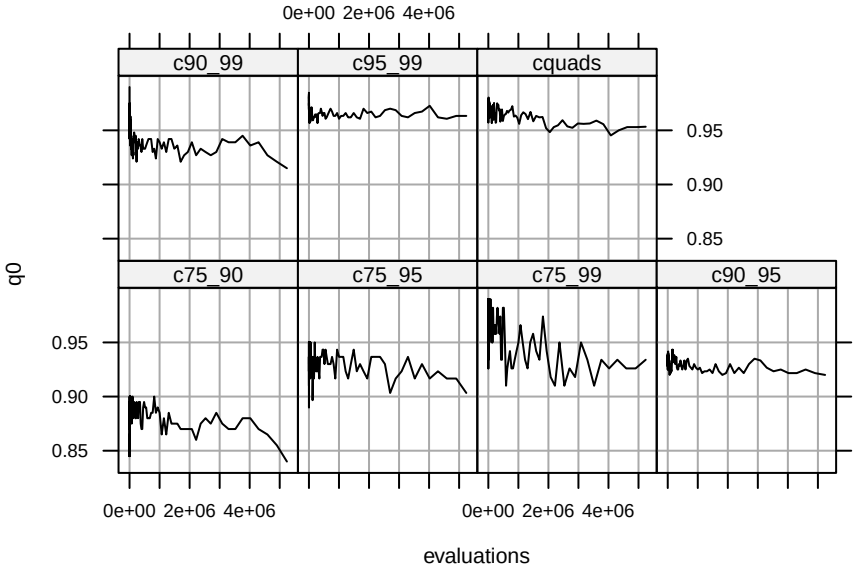

Fig. 10. q_0 values over time for an instance of size 783

Acknowledgments. This work was supported by Fundação para a Ciência e Tecnologia, under grant SFRH/PROTEC/67643/2010.

References

1. Birattari, M.: The Problem of Tuning Metageuristics from a machine learning perspective. Ph.D. thesis, Universite Libre de Bruxelles (December 2004)
2. Cai, Z., Huang, H., Qin, Y., Ma, X.: Ant colony optimization based on adaptive volatility rate of pheromone trail. Int. J. Communications, Network and System Sciences (2), 792–796 (2009)
3. Dorigo, M., Birattari, M., Stützle, T.: Ant colony optimization - artificial ants as a computational intelligence technique. Technical report, Université Libre de Bruxelles, Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (September 2006)
4. 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)
5. Dorigo, M., Maniezzo, V., Colorni, A.: Positive feedback as a search strategy. Tech. rep., Politecnico di Milano, Italy (1991)
6. Dorigo, M., Maniezzo, V., Colorni, A.: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics 26(1), 29–41 (1996)
7. Dorigo, M., Stützle, T.: Ant Colony Optimization. A Bradford Book, MIT Press, Cambridge Massachussetts (2004)

8. Favaretto, D., Moretti, E., Pellegrini, P.: Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, Lecture Notes in Computer Science, vol. 5752, chap. On the Explorative Behavior of MAX–MIN Ant System, pp. 115–119. Springer Berlin, Heidelberg (2009)
9. Forster, M., Bickel, B., Hardung, B., Kokai, G.: Self-adaptive ant colony optimisation applied to function allocation in vehicle networks. In: GECCO '07 Proceedings of the 9th annual conference on Genetic and evolutionary computation. pp. 1991–1998. ACM (2007)
10. Gaertner, D., Clark, K.: On optimal parameters for ant colony optimization algorithms. In: Arabnia HR, J.R. (ed.) Proceedings of the 2005 International Conference on Artificial Intelligence, ICAI 2005. pp. 83–89. CSREA Press (2005)
11. Goss, S., Aron, S., Deneubourg, J.L., Pasteels, J.M.: Self-organized shortcuts in the argentine ant. Naturwissenschaften 76, 579–581 (1989)
12. Hao, Z.F., Cai, R.C., Huang, H.: An adaptive parameter control strategy for aco. In: Proceedings of the Fifth International Conference on Machine Learning and Cybernetics. IEEE Press (2006)
13. Matthews, D.C., Sutton, A.M., Hains, D., Whitley, L.D.: Improved robustness through population variance in ant colony optimization. In: Stützle, T., Birattari, M., Hoos, H. (eds.) SLS '09 Proceedings of the Second International Workshop on Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics. pp. 145–149. Springer-Verlag Berlin Heidelberg (2009)
14. Melo, L., Pereira, F., Costa, E.: Mc-ant: a multi-colony ant algorithm. In: Collet, P., Monmarché, N., Legrand, P., Shoenauer, M., Lutton, E. (eds.) Artificial Evolution: 9th International Conference, Evolution Artificielle , EA'2009 LNCS 5975. pp. 25–36. Springer Berlin / Heidelberg (2009)
15. Pilat, M.L., White, T.: Using genetic algorithms to optimize acs-tsp. In: Dorigo, M., Caro, G.D., Sampels, M. (eds.) Ant Algorithms: Third International Workshop, ANTS 2002, LNCS, vol 2463. pp. 282–287. Springer, Heidelberg, Germany (2002)
16. Randall, M.: Near parameter free ant colony optimisation. In: Dorigo, M., Birattari, M., Blum, C., Gambardella, L., Mondada, F., Stützle, T. (eds.) ANTS'2004, Ant Colony Optimization and Swarm Intelligence, LNCS 3172. pp. 374–381. Springer-Verlag Berlin Heidelberg (2004)
17. Reinhelt, G.: TSPLIB: a library of sample instances for the tsp (and related problems) from various sources and of various types. URL: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
18. Ridge, E.: Design of experiments for the tuning of optimisation algorithms. Ph.D. thesis, Department of Computer Science, University of York, U.K. (2007)
19. 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)
20. Stützle, T., Lopez-Ibanez, M., Pellegrini, P., Maur, M., de Oca, M.M., Birattari, M., Dorigo, M.: Parameter adaptation in ant colony optimization. Technical report number tr/iridia/2010-002, IRIDIA, Bruxelles, Belgium (2010)
21. White, T., Pagurek, B., Oppacher, F.: Connection management using adaptive mobile agents. In: Arabnia, H. (ed.) Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98). pp. 802–809. CSREA Press (1998)
22. Zilberstein, S.: Using anytime algorithms in intelligent systems. AI MAGAZINE 17(3), 73–86 (1996)