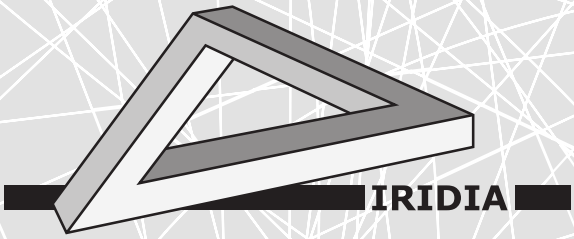

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Ant Colony Optimization on a Budget of 1000

L. Pérez Cáceres, M. López-Ibáñez, and T. Stützle

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2014-009

June 2014

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:
IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2014-009

Ant Colony Optimization on a Budget of 1000

Leslie Pérez Cáceres, Manuel López-Ibáñez, and Thomas Stützle

{leslie.perez.caceres, manuel.lopez-ibanez, stuetzle}@ulb.ac.be
IRIDIA, CoDE, Université libre de Bruxelles, Belgium

Abstract. Ant Colony Optimization (ACO) was originally developed as an algorithmic technique for tackling NP-hard combinatorial optimization problems. Most of the research on ACO has focused on algorithmic variants that obtain high-quality solutions when computation time allows the evaluation of a very large number of candidate solutions, often in the order of millions. However, in situations where the evaluation of solutions is very costly in computational terms, only a relatively small number of solutions can be evaluated within a reasonable time. This situation may arise, for example, when evaluation requires simulation. In such a situation, the current knowledge on the best ACO strategies and the range of the best settings for various ACO parameters may not be applicable anymore. In this paper, we start an investigation of how different ACO algorithms behave if they have available only a very limited number of solution evaluations, say, 1000. We show that, after tuning the parameter settings for this type of scenario, still the original Ant System performs relatively poor compared to other ACO strategies. However, the best parameter settings for such a small evaluation budget are very different from the standard recommendations available in the literature.

1 Introduction

The first Ant Colony Optimization (ACO) algorithms were introduced more than two decades ago [5]. After the publication of the main journal article describing Ant System (AS) [7], a large number of other ACO algorithms were introduced with the goal of improving over AS's performance and of showing that ACO algorithms could reach highly competitive results for various well-known combinatorial optimization problems. These improved algorithms include Ant Colony System (ACS) [6], Max-Min Ant System (MMAS) [21], rank-based Ant System (RAS) [4] and various others [8]. The main test problems at that time included the Traveling Salesman Problem (TSP) [4,6,7,21] and the Quadratic Assignment Problem (QAP) [7, 21], among few others.

When it comes to computational effort, typically a sufficiently large number of solutions constructed or significantly long computation times have been considered. For example, in the 1996 "First International Contest on Evolutionary Optimisation" [3], the competing algorithms could evaluate up to $10\,000 \cdot n$ candidate solutions, where n is the problem dimension, that is, the number of cities in the TSP. Similarly, in most papers large enough computation times have been given to allow a large number of solutions to be generated or expensive local

search methods to be used [9]. In particular, local search usually requires the evaluation of a large number of solutions.

On the other hand, there are situations where an algorithm can generate and evaluate only very few solutions before having to return the best solution found. This is the case when there are very tight real-time constraints even when it is quick to evaluate individual solutions or when the evaluation of solutions itself is very costly and in reasonable computation times only a small number of solutions can be evaluated. Common examples for the latter can be found in the field of simulation-optimization [1, 14, 23, 24]. Moreover, in such a situation the usage of incremental updates to explore neighboring candidate solutions, one of the key factors that make local search algorithms fast [10], is often not applicable. In such situations, an ACO algorithm may only be able to evaluate a few thousand (or even fewer) solutions.

When facing a situation where very few solutions can be evaluated, a first question is how to transfer the knowledge available in the ACO literature, since most experiments on ACO algorithms typically consider many more solution evaluations. Hence, we consider it a valid question to pose which algorithmic ACO variants may be the most promising in such situations and also which values their parameters should take. In this paper, we explore these questions, acknowledging that similar questions have been posed on single ACO algorithms but usually at still much higher computation budgets than we consider here [18]. We do so by comparing the performance of some of the main ACO algorithms, including AS, elitist AS (EAS), RAS, MMAS, and ACS using their default parameter values recommended in the literature [8] and using their parameters tuned by irace [13], an automatic algorithm configuration tool. As benchmark problems, we use the TSP and the QAP, but with the additional limitation that at most 1 000 candidate solutions can be evaluated per run. Our experimental results show that the ACO algorithms that were proposed as improvements over AS still are clearly preferable, even in such a scenario. However, for some of the ACO algorithms this is only the case after re-tuning their parameter settings. In fact, some of the tuned parameter settings differ very strongly from what has been recommended in the literature and, in some cases such as for ACS, from what intuition would dictate as a good setting.

The article is structured as follows. In Sec. 2 we give details on the algorithms we used, the parameter ranges considered and the benchmark problems we used for our tests. Sec. 3 gives the experimental results and we conclude in Sec. 4.

2 Experimental Setting

2.1 Problems

In this article, we consider a scenario where evaluating a solution is costly enough that only a few candidate solutions can be evaluated per run. To allow for a significant number of experiments and to allow for parameter tuning, we evaluate the ACO algorithms on two standard test problems (the TSP and the QAP), but

we restrict the number of solution evaluations to 1000. Both the TSP and the QAP are well-known NP-hard combinatorial optimization problems, often used as benchmarks for heuristic algorithms and in the early literature on ACO [8]. For both problems, we generate a set of benchmark instances to be used in the evaluation of the ACO algorithms.

For the TSP, we generate random uniform Euclidean instances, where points are randomly distributed in a square of dimension 10000×10000. We generate 100 instances for each value of 50, 60, 70, 80, 90, and 100 cities. Half of these instances of each size are used as training instances for the ACO algorithm tuning, while the other half are used as test set for the comparison of the algorithms. For the QAP, we use the instances proposed in [19]. These QAP instances have a structure analogous to the instances that arise in practical applications of the QAP. The instance set comprises 100 instances of each size 60, 80, and 100.

2.2 ACO Algorithms

In our experiments, we use five of the best-known ACO algorithms, namely AS [7], EAS [7], RAS [4], MMAS [21] and ACS [6]. A detailed description of the above algorithms can be found in [8]; here we recall just the main algorithmic rules in the solution construction and the pheromone update so that the parameters we later tune are defined.

ACO algorithms iteratively construct solutions to a problem by using heuristic information and pheromone trails. Most ACO algorithms make use of the random-proportional rule that was introduced with AS: At a decision point i, the next element j is chosen with a probability p_{ij} that is proportional to $\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}$, where τ_{ij} is the pheromone related to a choice of solution component (i, j), η_{ij} is the associated heuristic information and α and β are two parameters that weigh the influence of the pheromone with respect to the heuristic information. ACS used a more deterministic construction, where with a probability q_0, the next element j is chosen deterministically as the one that maximizes $\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}$ (ties being broken randomly). AS-based algorithms may also use this latter rule, that is, make a deterministic choice with probability q_0 and we consider this possibility also in this paper. Once all m ants have constructed a solution, where m is a parameter corresponding to the colony size, the pheromones are updated by evaporating a factor ρ of each pheromone trail, where ρ is a parameter ($0 \leq \rho < 1$), and depositing an amount of pheromone that is inversely proportional to the solution cost. The various ACO algorithms differ in which ants deposit pheromones and how much they deposit. For example, in AS each ant deposits an amount of pheromone equal to the inverse of the solution cost; in EAS the best solution since the start of the algorithm, the best-so-far solution, deposits additionally a large amount of pheromone; in RAS only some of the best solutions generated in each iteration and the best-so-far solution deposit pheromone; in MMAS only one ant deposits pheromone, which may be either the iteration-best or the best-so-far ant; finally, in ACS typically only the best solution since the start of the algorithm deposits pheromone. As a result, especially the more recent extensions such as RAS, MMAS, and ACS tend to exploit

better the best solutions found during the search, but possibly only after longer computation times [8].

For this paper, we have used the implementation of the ACO algorithms given by the ACOTSP software [20]. We do not use candidate sets or local search to avoid biases due to a priori exploitation of specific problem features. For the QAP, we have adapted the ACOTSP software in a straightforward way so that we could use the same implementation. The main difference between the ACO algorithms for the TSP and the QAP is that for the latter we have not derived heuristic information. In the TSP case the avoidance of heuristic information can simply be simulated by setting $\beta = 0$. We also extended ACOTSP such that parameter settings may vary during a single run as described in [15].

2.3 Automatic Configuration

We compare the ACO algorithms using default parameter settings, which were normally derived considering other application scenarios, such as rather large numbers of solution evaluations, and the ACO algorithms after tuning. As tuning tool we use the `irace` software [13] that implements Iterated F-race and other racing methods for automatic parameter tuning [2]. The details of the various configuration scenarios are described below and the scenario files are available as supplementary material (`http://iridia.ulb.ac.be/supp/IridiaSupp2014-006/`).

ACOTSP, ACOQAP: These configuration scenarios execute the algorithms using fixed parameter values and they consider a fixed maximum budget for the run of each algorithm of 1 000 evaluations. These scenarios require the configuration of five (QAP) or six (TSP) parameters common to all ACO algorithms, plus one specific parameter in the case of configuring EAS (elitistants: the weight given to the best-so-far solution) or RAS (rasrank: the maximum rank considered corresponding to the maximum number of ants (m) that deposit pheromone). The parameters and their ranges are given in Table 1. The tuning goal is to minimize the solution cost reached after 1 000 solution evaluations.

ACOTSP-V, ACOQAP-V: These scenarios allow the parameters to vary during the algorithm execution by tuning pre-scheduled parameter variations identical to those described in Table 2 of [15]; the other parameters use fixed settings in a range as indicated in Table 1. The pre-scheduled parameter variation is possible for the four parameters; m, β (only in the case of TSP), q_0 and ρ. The rationale for using this alternative tuning scenario is to examine whether parameter variations may improve performance. The configuration goal remains the same as in the ACOTSP and ACOQAP scenarios.

ACOTSP-VA, ACOQAP-VA: These scenarios consider the algorithms with the possibility of pre-scheduled parameter variations as in the previous scenario. However, they consider a different configuration goal: The optimization of the anytime behavior as measured by the normalized hypervolume of the space that is dominated by the pairs of points that describe the development of the best-so-far solution quality over the number of solution evaluations [15].

Table 1. Range of parameters used in the tuning for fixed parameter settings.

Common parameters for all scenarios				RAS and EAS		TSP scenario
m	α	ρ	q_0	$ras rank$	$elitistants$	β
$[5, 100]$	$[0, 10]$	$[0.01, 1]$	$[0, 1]$	$[1, 100]$	$[1, 175]$	$[0, 10]$

In this case, the configuration goal is to minimize the normalized hypervolume. The goal of this tuning setting is to obtain a parameter configuration that is good independent of the specific maximum number of solution evaluation that is given. To still optimize the algorithm behavior for short runs, the maximum execution budget of each run of an ACO algorithms was limited to 5 000 solution evaluations.

In all scenarios, the total configuration budget for each tuning run was 10 000 runs of the algorithm and, as said above, half of the available benchmark instances are used as training set for the tuning.

3 Experimental Results

In this section, we examine the ACO algorithms using different parameter settings. First, we compare the results of the five ACO algorithms when using their default settings in the ACOTSP software. Next, we consider parameter settings that have been tuned following scenarios ACOTSP and ACOQAP. Finally, we consider tuning parameter variation strategies and the anytime behavior of the ACO algorithms, that is, scenarios ACOTSP-V / VA and ACOQAP-V / VA. In the following, each time statistical significance tests are mentioned they refer to Wilcoxon rank-sum tests at the 0.05 significance level with Bonferroni's correction for multiple tests. The experimental results reported here are based on one run on each of the test instances (300 instances for the TSP and 150 instances for the QAP).

3.1 Default parameter settings

As a first step, we compare the five ACO algorithms using default parameter settings. For the presentation of the results, we use AS as a baseline, that is, we compute the relative quality deviation obtained by each ACO algorithm with respect to AS on each instance. More concretely, for each test instance i and each ACO algorithm a, we compute the percentage deviation of the result obtained by a on instance i from the result of AS on the same instance i. Figure 1 gives the boxplots of the resulting deviations. A value larger then zero indicates worse performance than AS, while a value lower than zero indicates better performance.

Maybe surprisingly, when using the default settings and limiting the number of evaluations to 1000, some of the ACO algorithms perform much worse than AS. This is particularly striking for MMAS, which generates tours that are about

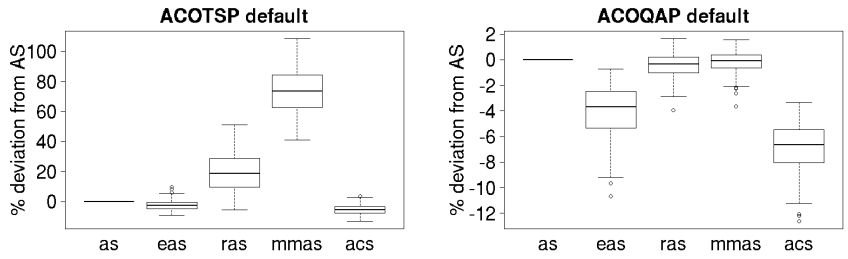

Fig. 1. Boxplots of the percentage deviation of the solution quality obtained by five ACO algorithms from the solution generated by AS, which is taken as reference. The results are given across all test instances.

70% worse than those of AS. Also RAS performs much worse than AS on the TSP. The poor performance of these algorithms for short runs is due to the fact the MMAS and RAS parameters were set to allow for a very high final solution quality after a large number of candidate solutions have been generated [8]; for example, both MMAS and RAS use a relatively small evaporation, which does not allow them to bias the search fast enough to focus on the best solutions found. Another reason is the different default setting of the parameter β, which for AS is set to 5 whereas for MMAS and RAS is set to 2. EAS and ACS both show better performance than AS. All differences are statistically significant at the 0.05 significance level. For the QAP, the situation is different from the one of the TSP. None of the other ACO algorithms performs worse than AS. Considering statistical significance, EAS, RAS and ACS perform statistically significantly better than AS, whereas there is no statistically significant difference between MMAS and AS. This difference to the TSP results can be explained by the fact that the ACO algorithms for the QAP do not make use of heuristic information. In fact, if one eliminates heuristic information for the TSP by setting $\beta = 0$, the performance relative to AS follows the same trends as for the QAP (more details are given in the supplementary material).

3.2 Tuned settings

In a next step, we tuned the parameter settings for both ACOTSP and ACOQAP scenarios as defined in Section 2.3. After tuning, all algorithms significantly improve the solution quality reached within the limit of 1 000 candidate solutions. Figure 2 shows the relative deviation of each tuned ACO algorithm over its default version. As before, negative values indicate improved quality. The algorithms that most improve their performance are MMAS and RAS, while AS on the QAP is the only ACO algorithm that does not strongly improve its quality after tuning.

Figure 3 compares the performance of the ACO algorithms using AS as a reference in the same way as in the previous section, that is, the relative deviation of the quality obtained by the tuned version of each ACO algorithm with respect

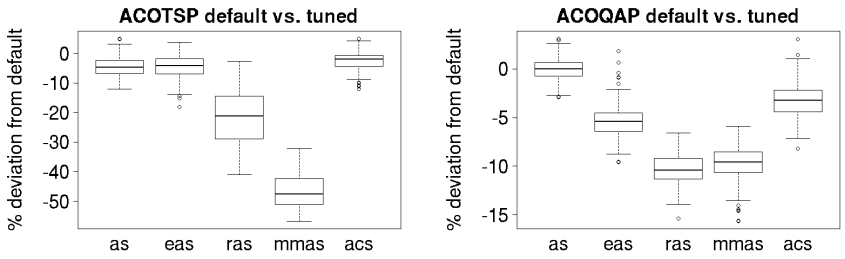

Fig. 2. Boxplots of the observed percentage improvement of each ACO algorithm with tuned parameter settings over the solutions reached with default parameter settings. The reference cost of each tuned ACO algorithm is its respective default parameter setting. For example, the boxplot of MMAS indicates the improvement observed of MMAS with tuned parameter settings over its default parameter settings. The results are given across all test instances.

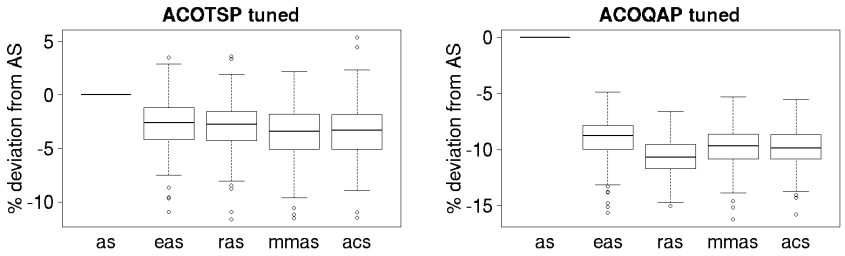

Fig. 3. Boxplots of the percentage deviation of the solution quality obtained by five ACO algorithms from the solution generated by AS, which is taken as reference. The results are given across all test instances.

to the tuned version of AS. This comparison shows that, for both the TSP and QAP, the four ACO algorithms (EAS, RAS, MMAS, and ACS) improve significantly over the performance of AS. (All the differences between AS and the other ACO algorithms are statistically significant.) The overall best performance on the TSP and on the QAP is obtained by MMAS and RAS, respectively.

A main reason for the strong improvements of most ACO algorithms over their default parameter settings is that these settings were designed for scenarios where ample computation time is available, that is, where a large number of candidate solutions may be constructed. To examine the differences between the default and the tuned parameter settings, we performed for each ACO algorithm 20 runs of irace and we analyzed the distribution of the parameter configurations that were obtained. These distributions are given in Figure 4 using boxplots; the red line for each algorithm indicates the default parameter settings.

While the parameter settings selected by irace varied from run to run, we can observe some clear trends. The algorithm for which the tuned parameter settings differ the least from the default ones is AS. For the other algorithms,

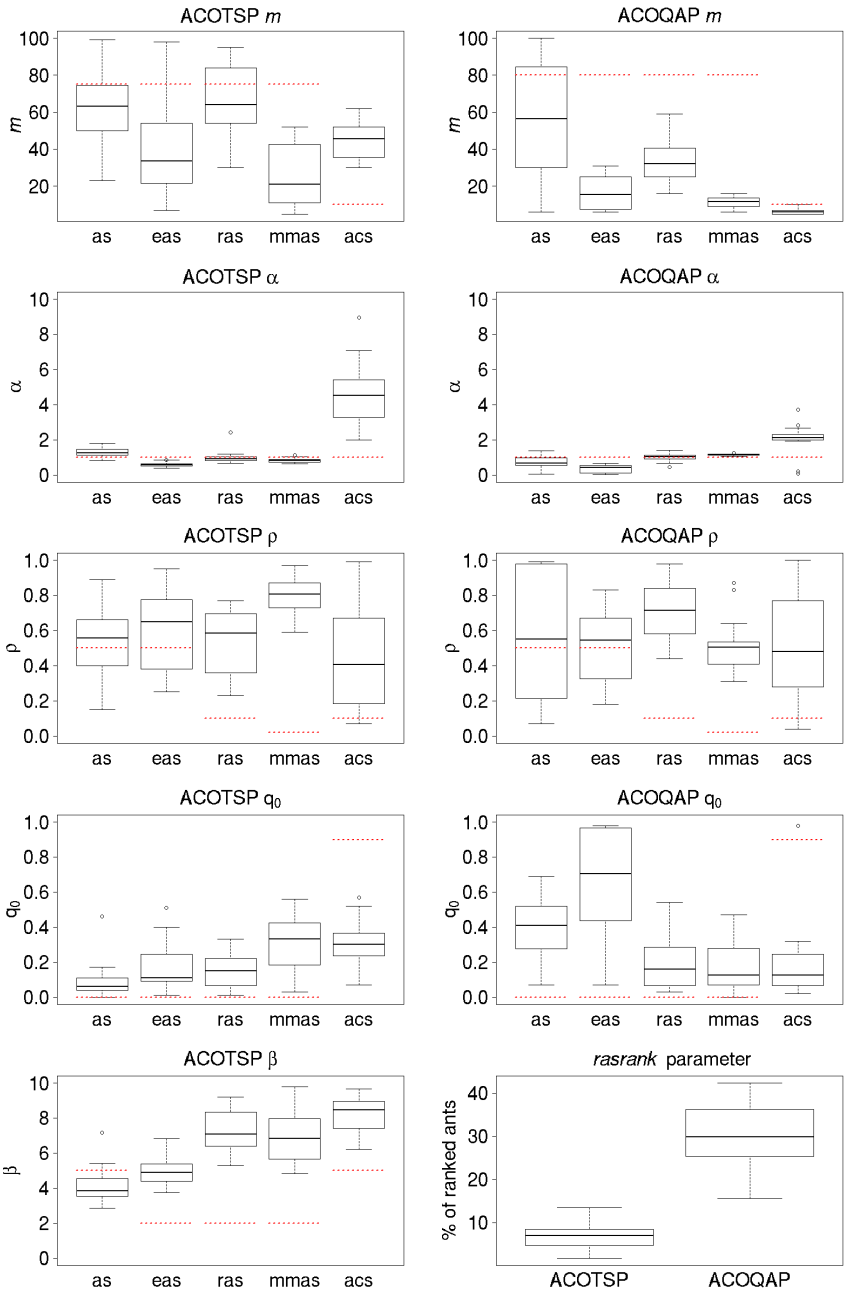

Fig. 4. Distribution of the parameter values found in 20 runs of irace. The dotted red lines indicate the default parameter setting for each algorithm. For the number of ants, the default parameter setting is the instance size; since in our test instances the size varies, we assumed an "average" instance size of 80 to indicate the default setting.

major differences arise. RAS and especially MMAS use much smaller number of ants (m) than in their default version, which allows them to perform more iterations than in the default settings. In the ACOTSP scenario, also the heuristic information gets a much higher emphasis by using a median value for β of around seven instead of the default setting of two. The most noteworthy is probably the much higher evaporation rate ρ in RAS and MMAS than the default evaporation rate for both the ACOTSP and the ACOQAP scenarios. A very high pheromone evaporation has the effect that the search can quickly forget previously obtained worse solutions and focus quickly around the best recent ones. All these differences can be explained by the need of exploiting much more aggressively the search history (and heuristic information if available and helpful) due to the very small number of solutions to be generated biasing in this way the search around the best solutions found so far.

The setting of q_0 larger than zero for all ACO algorithms supports this interpretation, although in the case of ACS the tuned value for q_0 is somewhat surprising: a rather low value of q_0 together with a rather high value for α is proposed by the tuning, thus providing here another means for the exploitation of the search history. If we compare the settings of the ACOTSP and the ACOQAP scenarios, we can observe similar overall trends. However, there are differences in the best parameter settings for specific algorithms. For example, differences in good parameter settings in the two scenarios are evidenced by the fact that the boxplots of the distribution of the tuned parameter settings for the same ACO algorithm in the ACOTSP and ACOQAP scenarios often do not overlap.

3.3 Parameter variation and anytime parameter tuning

Instead of using fixed parameter settings, another option may be to adapt the parameter settings while running the algorithm. Earlier studies have indicated that pre-scheduled parameter variations [16, 22] may be more promising than self-adaptive schemes for deriving improved ACO parameter settings [17]. In particular, pre-scheduled parameter variation was shown to be particularly successful to improve the anytime behavior of MMAS [15]. In this section, we vary the parameters q_0, β, ρ, and m within a single run using the same variation schemes as proposed in [15]: We consider for all parameters as possible changes either their gradual variation iteration-by-iteration or a single switch of the parameter setting from some value to another one at a particular iteration.

For tuning, we consider the two possibilities that are offered by the scenarios ACOTSP-V / VA and ACOQAP-V / VA, which were described in Section 2.3. In a nutshell, the results do not show a strong difference when tuning for the final quality or for the anytime behavior, when the algorithms are evaluated according to the quality reached at 1 000 evaluations (Fig. 5).

On the other hand, the possibility of tuning the anytime behavior has a positive impact on the behavior of at least some of the algorithms if we look at the solution quality reached for different values of the evaluation budget. The plots of the solution quality development (as measured by the percentage deviation of the algorithms from the AS solutions) over the number of evaluations (Fig. 6)

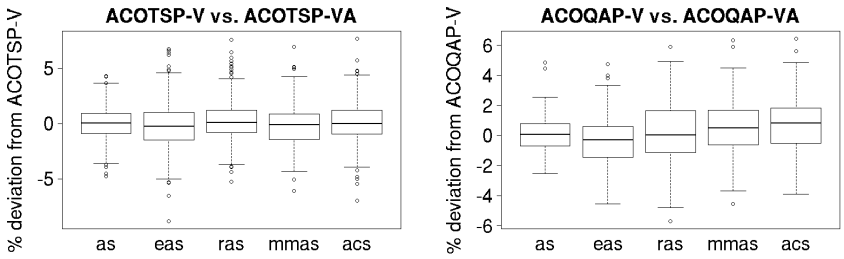

Fig. 5. Performance the ACO algorithm tuned for optimizing their anytime behavior versus the fixed parameter settings tuned for best performance at 1 000 evaluations. The results are given for an evaluation of the algorithms on the test set after 1 000 evaluated candidate solutions.

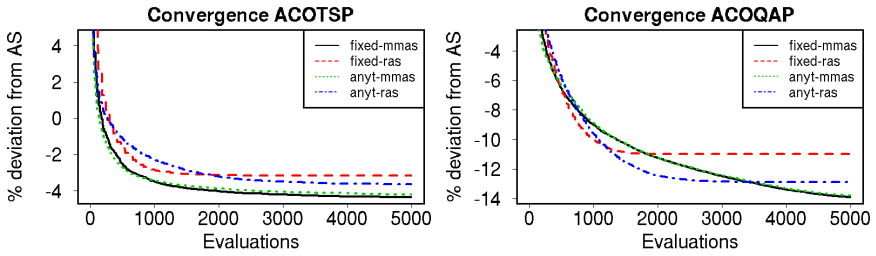

Fig. 6. Plots of the development of the solution quality over the number of solution evaluations for the TSP (left) and the QAP (right). The solution quality is measured as the percentage improvement over the solutions generated by AS across the test instances. The algorithms shown (MMAS and RAS) where tuned either for a fixed evaluation budget (fixed) and for optimizing their anytime behavior (anyt).

show that, while MMAS' curve is almost identical independently of how the tuning was done, the curves for RAS show a clear stagnation effect of the RAS tuned for a fixed evaluation budget when compared to the version of RAS tuned for anytime behavior. Since it may be unknown a priori how many evaluations can be done in practice, we would recommend tuning for anytime behavior.

4 Final Remarks and Future Work

In this paper, we have analyzed the performance of five ACO algorithms for very low budgets on the evaluation of candidate solutions. Our computational results showed that EAS, RAS, MMAS, and ACS improve in performance over AS even in such circumstances. However, to make these algorithms reach high performance in short runs, very different parameter settings from the usual default ones have to be used.

There are a number of directions in which this work can be extended. Even though by the analysis of the tuned parameter settings we obtained new insights

into the best parameter values for very short runs, the current tuning setting is maybe not the most realistic one. In fact, in an expensive function evaluation setting, the tuning would be rather time-consuming and it may be unrealistic to afford a time-intensive fine-tuning for each different problem being tackled. A next step would be to obtain more general settings. For example, we may tune the ACO algorithms across many different combinatorial problems and in this way derive robust parameter settings. Another question concerns whether to use specific known ACO algorithms or rather consider a framework of ACO algorithms from which known and new ACO algorithms may be instantiated. Such a framework may lead ultimately to a higher performing ACO algorithm than those we know nowadays–and this also holds for the here considered settings of very few solution evaluations. Finally, another possibility is to use surrogate modeling approaches to model fitness landscapes [11, 12]. While such surrogate modeling approaches have widely been applied to black-box continuous function optimization problems, their usage is more rare for combinatorial optimization problems. Hence, the exploration of such models and their integration into ACO algorithms may be a promising next step.

Acknowledgments. This work received support from the COMEX project within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office. Manuel López-Ibáñez and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS, of which they are a postdoctoral researcher and a senior research associate, respectively. Leslie Pérez Cáceres acknowledge support of CONICYT Becas Chile.

References

1. April, J., Glover, F., Kelly, J., Laguna, M.: Practical introduction to simulation optimization. In: Simulation Conference, 2003. Proceedings of the 2003 Winter. vol. 1, pp. 71–78 (Dec 2003)
2. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein, T., Blesa, M.J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., Sampels, M. (eds.) HM'07', LNCS, vol. 4771, pp. 108–122. Springer (2007)
3. Bersini, H., Dorigo, M., Langerman, S., Seront, G., Gambardella, L.M.: Results of the first international contest on evolutionary optimisation. In: Bäck, T., Fukuda, T., Michalewicz, Z. (eds.) Proceedings of ICEC'96. pp. 611–615. IEEE Press, Piscataway, NJ (1996)
4. Bullnheimer, B., Hartl, R., Strauss, C.: A new rank-based version of the Ant System: A computational study. Central European Journal for Operations Research and Economics 7(1), 25–38 (1999)
5. Dorigo, M.: Optimization, Learning and Natural Algorithms. Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy (1992), in Italian
6. 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)

7. 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)
8. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge, MA (2004)
9. Gambardella, L.M., Montemanni, R., Weyland, D.: Coupling ant colony systems with strong local searches. European Journal of Operational Research 220(3), 831–843 (2012)
10. Hoos, H.H., Stützle, T.: Stochastic Local Search—Foundations and Applications. Morgan Kaufmann Publishers, San Francisco, CA (2005)
11. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global Optimization 13(4), 455–492 (1998)
12. Knowles, J.D., Corne, D., Reynolds, A.P.: Noisy multiobjective optimization on a budget of 250 evaluations. In: Ehrgott, M., Fonseca, C.M., Gandibleux, X., Hao, J.K., Sevaux, M. (eds.) EMO 2009, LNCS, vol. 5467, pp. 36–50. Springer (2009)
13. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011), `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf`
14. López-Ibáñez, M., Prasad, T.D., Paechter, B.: Ant colony optimisation for the optimal control of pumps in water distribution networks. Journal of Water Resources Planning and Management, ASCE 134(4), 337–346 (2008)
15. López-Ibáñez, M., Stützle, T.: Automatically improving the anytime behaviour of optimisation algorithms. European Journal of Operational Research 235(3), 569–582 (2014)
16. Maur, M., López-Ibáñez, M., Stützle, T.: Pre-scheduled and adaptive parameter variation in $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. In: Ishibuchi, H., et al. (eds.) Proceedings of CEC 2010, pp. 3823–3830. IEEE Press, Piscataway, NJ (2010)
17. Pellegrini, P., Birattari, M., Stützle, T.: A critical analysis of parameter adaptation in ant colony optimization. Swarm Intelligence 6(1), 23–48 (2012)
18. Pellegrini, P., Favaretto, D., Moretti, E.: On $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System's parameters. In: Dorigo, M., et al. (eds.) ANTS 2006, LNCS, vol. 4150, pp. 203–214. Springer (2006)
19. Pellegrini, P., Mascia, F., Stützle, T., Birattari, M.: On the sensitivity of reactive tabu search to its meta-parameters. Soft Computing (In press)
20. Stützle, T.: ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem (2002), `http://www.aco-metaheuristic.org/aco-code/`
21. Stützle, T., Hoos, H.H.: $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. Future Generation Computer Systems 16(8), 889–914 (2000)
22. Stützle, T., López-Ibáñez, M., Pellegrini, P., Maur, M., Montes de Oca, M.A., Birattari, M., Dorigo, M.: Parameter adaptation in ant colony optimization. In: Hamadi, Y., Monfroy, E., Saubion, F. (eds.) Autonomous Search, pp. 191–215. Springer, Berlin, Germany (2012)
23. Teixeira, C., Covas, J., Stützle, T., Gaspar-Cunha, A.: Multi-objective ant colony optimization for solving the twin-screw extrusion configuration problem. Engineering Optimization 44(3), 351–371 (2012)
24. Zeng, Q., Yang, Z.: Integrating simulation and optimization to schedule loading operations in container terminals. Computers & Operations Research 36(6), 1935–1944 (2009)