Off-line and On-line Tuning: A Study on Operator Selection for a Memetic Algorithm Applied to the QAP

Gianpiero Francesca[1], Paola Pellegrini[2], Thomas Stützle[2], and Mauro Birattari[2]

[1] Dipartimento di Ingegneria, University of Sannio, Benevento, Italy
gianpiero.francesca@gmail.com
[2] IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
paolap@pellegrini.it, {stuetzle,mbiro}@ulb.ac.be

Abstract. Tuning methods for selecting appropriate parameter configurations of optimization algorithms have been the object of several recent studies. The selection of the appropriate configuration may strongly impact on the performance of evolutionary algorithms. In this paper, we study the performance of three memetic algorithms for the quadratic assignment problem when their parameters are tuned either off-line or on-line. Off-line tuning selects *a priori* one configuration to be used throughout the whole run for all the instances to be tackled. On-line tuning selects the configuration during the solution process, adapting parameter settings on an instance-per-instance basis, and possibly to each phase of the search. The results suggest that off-line tuning achieves a better performance than on-line tuning.

1 Introduction

Tuning an algorithm means to select its configuration, that is, a specific setting of all relevant parameters. The selection of the appropriate configuration has a major impact on the performance of evolutionary algorithms and, more generally, of all stochastic optimization algorithms. Several automatic tuning methods are available in the literature.

Tuning methods can be grouped in two main categories, namely off-line and on-line ones. In off-line methods the configuration to be used is selected after testing several ones on a set of tuning instances. The selected configuration is then used for solving all instances to be tackled. Off-line methods typically consider the algorithm to be tuned as a black-box. Thus, they may be easily applied to any algorithm without any intervention on the algorithm itself [1,2,3,4]. On-line methods vary the configuration during the solution of the instances to be tackled, by exploiting some feedback from the search process [5,6,7]. On-line tuning is often named parameter control, or parameter adaptation [8,9].

In this paper, we study the performance achieved by three memetic algorithms for the quadratic assignment problem, when the crossover operator [7] is

P. Merz and J.-K. Hao (Eds.): EvoCOP 2011, LNCS 6622, pp. 203–214, 2011.
© Springer-Verlag Berlin Heidelberg 2011

tuned either off-line or on-line. We focus here on the operator selection since it is recognized to be a major issue when dealing with evolutionary algorithms, as it has a great impact on the performance achieved [10]. We test one off-line and three on-line methods for selecting the appropriate operator out of a set of four possible ones. The configuration space so obtained is very small, and, thus, the tuning problem can be considered rather simple. In fact, we earlier have shown that the increase of the dimension of the configuration space penalizes on-line more than off-line tuning [11]. Thus, the current experimental setup can be seen as the most favorable for on-line tuning. We compare the tuning methods as a function of the quality of the specific algorithm to be tuned. These quality differences are obtained by considering variants of the memetic algorithm. Our initial conjecture was that the performance level of an algorithm may have an impact on the relative desirability of off-line *vs.* on-line tuning methods. Therefore, we tested the tuning methods on three variants of the memetic algorithm: the first variant does not include either local search or mutation operator; the second one includes local search, but no mutation operator; the third one includes both, local search and the mutation operator.

The results obtained are actually not fully conclusive: only some trends can be detected for supporting our initial conjecture. In general, off-line tuning is the best performing method, with on-line tuning achieving seldomly the best results. Still, some relation may exist between the method to be preferred and the quality of the algorithm. In particular, one should prefer off-line tuning when a high quality algorithm is to be applied. Surprisingly, the heterogeneity of the instances to be solved does not have a remarkable impact on the results.

The rest of the paper is organized as follows: in Section 2, we describe the memetic algorithms we consider in this study, in Section 3, we present the tuning methods we apply. In Section 4, we depict the experimental setup, and in Section 5, we discuss the results obtained. In Section 6, we draw some conclusions.

2 The Algorithms Implemented

Memetic algorithms (MA) represent one of the most successful approaches in the field of evolutionary computation [12]. Typically, a memetic algorithm combines a population based technique and a local search.

In the experimental analysis reported in this paper, we tackle the quadratic assignment problem (QAP). In the QAP, a set of n facilities are to be assigned to a set of n locations. A flow f_{ij} is associated to each pair of facilities $i, j = 1, ..., n$, and a distance d_{hk} is given for each pair of locations $h, k = 1, ..., n$. A solution of the QAP is an assignment of each facility to a location, and it can be represented as a permutation π: the value in the $i-$th position of the permutation, $\pi(i)$, corresponds to the facility that is assigned to the $i-$th location. The cost of a solution is equal to the sum over all pairs of facilities of the product of the flow between them, and the distance between their assigned location:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} f_{\pi(i)\pi(j)}d_{ij}.$$

The goal of the QAP is to find the solution that minimizes the cost of the assignment.

In MA, each individual represents a solution of the problem. In the initialization phase of our MA for the QAP, a population of p individuals is randomly generated and it is improved by local search. The algorithm evolves the current population through crossover and mutation operators, until a stopping criterion is fulfilled. At each iteration, the algorithm generates p_c new individuals through a crossover operator. A crossover operator generates an individual by combining two different ones belonging to the current population. The new individual is named offspring, the two preexisting ones are named parents. A mutation operator modifies an individual. After crossover and mutation, local search is applied to each individual. The new population is obtained by selecting the best p individuals from both old and new ones. For avoiding premature convergence, the search is restarted as soon as the average distance between individuals becomes smaller than a predefined threshold t. In this case, the new population is generated randomly.

We study the performance of three algorithms that are inspired by the implementation proposed by Merz and Freisleben [13]. They differ in the application of either the local search or the mutation operator. The first algorithm (simple MA) does not adopt either local search or a mutation operator (actually, this is not really an MA, but we keep this name for simplicity of language). The second algorithm (intermediate MA) adopts local search, but it does not adopt a mutation operator. The third algorithm (full MA) adopts both local search and a mutation operator. The mutation operator performs a random perturbation of individuals. In particular, the algorithm randomly draws a number of $p_m = p/2$ individuals from the overall population, including both the p current individuals and the new ones generated through crossover. For each individual, the operator iteratively exchanges elements in the permutation selecting them randomly according to a uniform distribution. Such exchanges are performed until the distance between the original and the resulting individuals is higher than a predefined threshold m. The distance between two individuals is equal to the number of components with different values.

A crossover operator generates an offspring, I_o, starting from a pair of parents, I_{p_1} and I_{p_2}. We consider the crossover operator to be used as a parameter with four possible settings.

The **cycle crossover** operator, CX [14], copies to the offspring all components that are equal in both parents. The remaining components of I_o are assigned starting from a random one, $I_o(j)$, according to the following procedure. One of the two parents is randomly drawn. Let it be I_{p_1}. CX sets $I_o(j) = I_{p_1}(j)$. Then, let $I_{p_1}(j')$ be the component such that $I_{p_1}(j') = I_{p_2}(j)$: CX sets $I_o(j') = I_{p_1}(j')$, and it substitutes the index j with j'. This procedure is repeated until all components of I_o are instantiated.

The **distance preserving crossover**, DPX [13,15], generates an offspring that has the same distance from both parents. DPX copies in I_o all the components that are equal in I_{p_1} and I_{p_2}. Each remaining component $I_o(j)$ is randomly

assigned, provided that $I_o(j)$ is a permutation and it is different from both $I_{p_1}(j)$ and $I_{p_2}(j)$.

The **partially mapped crossover** operator, PMX [16], randomly draws two components of I_o, $I_o(j)$ and $I_o(j')$, $j < j'$. It sets $I_o(k) = I_{p_1}(k)$ for all $k < j$ or $k > j'$, and $I_o(k) = I_{p_2}(k)$ for all $j \leq k \leq j'$. If the so obtained offspring is not a feasible solution, for each pair of components $I_o(k)$ and $I_o(z)$ such that $I_o(k) = I_o(z)$, $j \leq z \leq j'$, PMX sets $I_o(k) = I_{p_1}(k)$.

The **order crossover**, OX [17], randomly draws two components of I_o, $I_o(j)$ and $I_o(j')$. It sets $I_o(k) = I_{p_1}(k)$ for all $j \leq k \leq j'$. Then, OX copies in the k^{th} unassigned component of I_o the k^{th} component of I_{p_2} that differs from any $I_o(z)$, $j \leq z \leq j'$.

3 Parameter Tuning

For selecting the appropriate configuration of the three MAs described in Section 2, we apply one off-line and three on-line tuning methods.

The off-line method performs an exhaustive exploration of the configuration space, based on a set of instances with characteristics that are similar to those of the instances to be tackled: all tuning instances are solved using all possible configurations in 10 independent runs.

The three on-line methods select the configuration to be used among the possible ones. The selection is a function of the quality of solutions previously generated by applying each configuration. The configuration to be used varies at each step, where a step corresponds to the generation of one offspring starting from two parents. The quality of a configuration c, Q_c, is evaluated after each iteration. The equation used for updating Q_c depends on a reward function R_c, which is given by

$$R_c = \frac{1}{|\mathcal{I}^c|} \sum_{I_o \in \mathcal{I}^c} \frac{f_{I_o}}{f_{I_{best}}} \max\left\{0, \frac{f_{I_o} - f_{I_p}}{f_{I_p}}\right\}, \tag{1}$$

where $\mathcal{I}^c$ is the set of offspring generated in the current iteration by configuration c; f_I is the value of the fitness function associated to individual I; I_{best} is the individual with the highest fitness generated up to the current iteration; I_p is the I_o's parent with the highest fitness. The contribution of each offspring to the reward is the product of two quantities. The first quantity is the ratio between the fitness of I_o and the one of I_{best}. The second quantity is the relative fitness improvement of I_o with respect to I_p, or zero in absence of an improvement.

In the first on-line method, named **probability matching**, PM, the selection of the configuration to be used is stochastic [18]. The quality Q_c associated to configuration c is updated as:

$$Q_c = Q_c + \max\{0, \alpha\,(R_c - Q_c)\}, \tag{2}$$

where α is a parameter of the algorithm, $0 < \alpha \leq 1$. The probability of selecting configuration c is:

$$P_c = P_{min} + (1 - |C|P_{min})\frac{Q_c}{\sum_{c' \in C} Q_{c'}}, \tag{3}$$

where C is the set of all possible configurations, and P_{min} is a parameter of the algorithm, $0 \leq P_{min} \leq 1$. In the initialization phase, the quality is initialized to $Q_c = 1$ for each configuration c, and the probability is uniformly distributed.

In the second on-line method, named **adaptive pursuit**, AP, as in probability matching, the selection of the configuration to be used is stochastic, and the probability distribution is based on a quality measure that is updated following Equation (2) [19]. Here, the probability of selecting configuration c, P_c, is computed as:

$$P_c = \begin{cases} P_c + \beta(P_{max} - P_c), & \text{if } Q_c = \max_{c' \in C}\{Q_{c'}\}, \\ P_c + \beta(P_{min} - P_c), & \text{otherwise,} \end{cases} \tag{4}$$

where C is the set of all possible configurations, P_{min} and β are parameters of the algorithm, $0 < \beta \leq 1$ and $0 \leq P_{min} \leq 1$, and P_{max} is set to $1-(|C|-1)P_{min}$.

In the third on-line method, named **multi-armed bandit**, MAB, the selection of the configuration to be used is deterministic [20]. The quality Q_c is computed as the average value returned by the reward function in all the iterations performed. The configuration selected $\bar{c}$ is:

$$\bar{c} = \arg\max_{c \in C} \left\{ Q_c + \gamma \sqrt{\frac{2 \ln \sum_{c' \in C} n_{c'}}{n_c}} \right\}, \tag{5}$$

where n_c is the number of offspring generated by using configuration c in all the iterations performed, and γ, $\gamma > 0$, is a parameter of the algorithm.

4 Experimental Setup

In the experimental analysis we study the performance of off-line and on-line tuned versions of three MA algorithms. By studying the various algorithms described in Section 2, we compare the performance achieved by the different tuning methods as a function of the quality of the algorithm. In addition, we analyze the performance of the algorithms when different values of CPU time are imposed as stopping criterion. We run the algorithms with the following default parameter settings: $p = 40$, $p_c = p/2$, $t = 30\%$; in full MA, $p_m = p/2$, $m = 40\%$. In intermediate and full MA, we apply the 2-opt local search with best improvement [21].

For each algorithm, we test seven different versions depending on the configuration selection policy:

- The configuration is maintained constant throughout the whole run: the configuration to be used is i) *default*, D: CX crossover operator; ii) *off-line*, OFF: the one selected by the off-line method; iii) *random*, R: random selection of one crossover operator according to a uniform probability distribution.

Table 1. Configuration selected by off-line tuning on the two sets of instances tackled, for each algorithm and for each CPU time limit in seconds

	homogeneous			heterogeneous		
time	10	31	100	10	31	100
simple	OX	PMX	PMX	PMX	PMX	PMX
intermediate	CX	CX	CX	CX	PMX	PMX
full	CX	PMX	PMX	CX	PMX	PMX

– The configuration to be used is selected at each step: iv) *naive*, N: random selection of the configuration, according to a uniform probability distribution; v) *probability matching*, PM: $\alpha = 0.3$ and $P_{min} = 0.05$ [18]; vi) *adaptive pursuit*, AP: $\alpha = 0.3$, $\beta = 0.3$ and $P_{min} = 0.05$ [19]; vii) *multi-armed bandit*, MAB: $\gamma = 1$ [20].

For a fair comparison between off-line and on-line tuning, all the methods select the configuration to be used from the same set of possibilities: the crossover operator can be set to CX, DPX, PMX, or OX.

We consider two sets of instances. First, we solve instances of size 50 to 100 from the QAPLIB [22]. We name these instances heterogeneous, since they come from very different backgrounds, they have different sizes, and they are either structured or unstructured. Second, we consider a set of instances obtained through the instance generator described by Stützle and Fernandes [23]. We name these instances homogeneous, since they are all unstructured, they have all size 80, and they are generated based on the same distributions. Both sets include 34 instances. We randomly split each set in two subsets. We use one of them for performing the off-line tuning. Table 1 reports the configuration selected by off-line tuning for each algorithm and for each CPU time limit, namely 10, 31 and 100 CPU seconds. In Section 5 we discuss the results achieved on the instances of the second subsets by the seven versions implemented. For the different stopping criteria, we perform 10 independent runs of each version on all instances.

All the experiments are performed on Xeon E5410 quad core 2.33GHz processors with 2x6 MB L2-Cache and 8 GB RAM, running under the Linux Rocks Cluster Distribution. The algorithms are implemented in C++, and the code is compiled using gcc 4.1.2.

5 Experimental Results

By analyzing the performance of three MA algorithms, we can observe the relative performance of the tuning methods as a function of the algorithm performance. Table 2 shows the percentage error with respect to the best known solution of each instance obtained by the default version of the three algorithms. We present the results obtained in one run of 100 seconds on both the homogeneous and heterogeneous instances. The best algorithm is the full one, followed

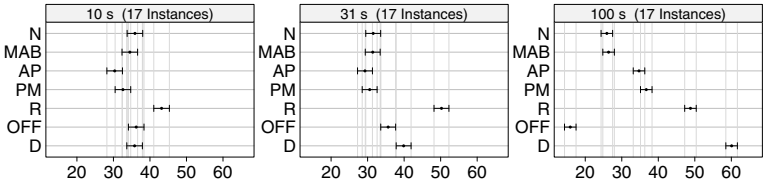
Table 2. Algorithm quality. Percentage error obtained after 100 seconds by the default version of the three MA algorithms, with respect to the best known solution of each instance.

	homogeneous	heterogeneous
simple	4.69343%	9.29772%
intermediate	1.51216%	2.17695%
full	0.79046%	1.44571%

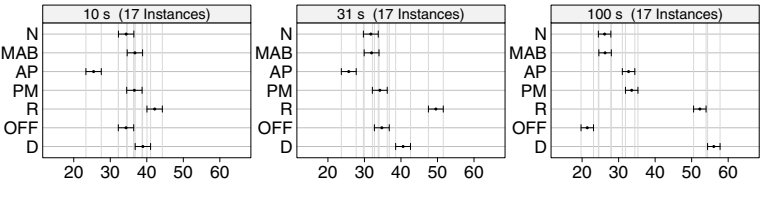
by the intermediate. The simple algorithm is the worst performing. For each instance set, the difference between all pairs of algorithms is statistically significant at the 95% confidence level, according to the Wilcoxon rank-sum test.

Simple MA. For assessing the performance of the seven versions of the simple algorithm as a function of different run-lengths on both heterogeneous and homogeneous instances, we present the results achieved in terms of ranking. We test the significance of the differences with the Friedman test for all-pairwise comparisons. The plots depicted describe the 95% simultaneous confidence intervals of these comparisons. For each version we show the median rank over all instances, together with the bounds of the corresponding confidence interval. If the intervals of two versions overlap, then the difference among these versions is not statistically significant [24]. We use the same type of representation for all results provided in the paper.

The results achieved on the homogeneous instances are reported in Figure 1(a). The off-line version performs significantly worse than at least one on-line version

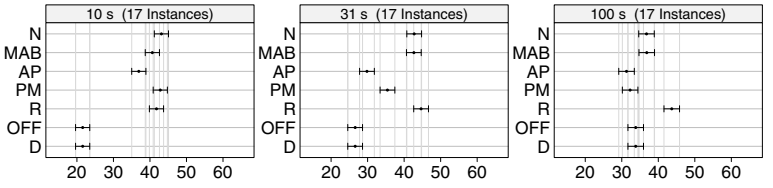


(a) homogeneous set

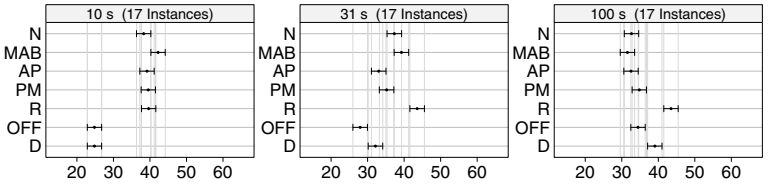


(b) heterogeneous set

Fig. 1. Results achieved by the seven versions of the simple algorithm. Simultaneous confidence intervals for all-pairwise comparisons of ranks between all versions applied to homogeneous and heterogeneous instances.



(a) homogeneous set



(b) heterogeneous set

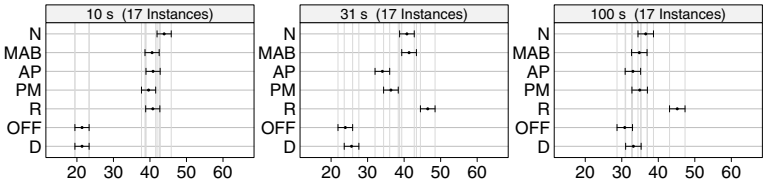
Fig. 2. Results achieved by the seven versions of the intermediate algorithm. Simultaneous confidence intervals for all-pairwise comparisons of ranks between all versions applied to homogeneous and heterogeneous instances.

for runs of 10 and 31 seconds, while it is the best one for the longest run-length. Which is the best on-line version depends on the CPU time available, even if in most cases the difference among these versions is not significant. For what concerns the benchmark versions, the random version outperforms only the default versions for runs of 100 seconds, and it is the worst performing otherwise. The naive version, instead, achieves quite good results. In particular, it is comparable to the best on-line method for medium and long runs.

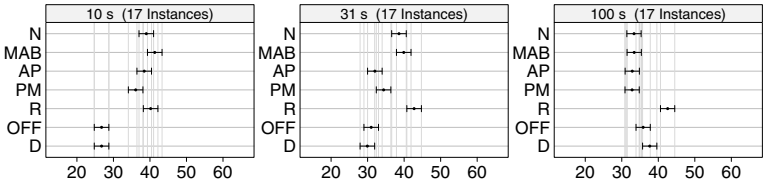
In Figure 1(b), we depict the results achieved on the heterogeneous instances. The qualitative conclusions that can be drawn are equivalent to those derived from the homogeneous instances. The off-line version is significantly worse than the best on-line version for the short and medium run-lengths, while the opposite holds for long ones. Which is the best on-line method depends on the run-length. On these instances, the difference between the best on-line version and the other ones is statistically significant for runs of 10 and 31 seconds.

Intermediate MA. In Figure 2(a), we report the results achieved on the homogeneous instances. The off-line version outperforms the best on-line one for the short run-length. They are comparable for runs of 31 and 100 seconds. Adaptive pursuit is the best on-line version for runs of 31 seconds. Naive and multi-armed bandit achieve very similar results, and they outperform only the random version. Differently from the case of the simple algorithm, the default version achieves good results: it is always statistically equivalent to the best version.

The results on the heterogeneous instances, reported in Figure 2(b), show that the characteristics of the instances do not have a remarkable impact on the



(a) homogeneous set



(b) heterogeneous set

Fig. 3. Results achieved by the seven versions of the full algorithm. Simultaneous confidence intervals for all-pairwise comparisons of ranks between all versions applied to homogeneous and heterogeneous instances.

relative performance of off-line tuning: the off-line version is the best one for the short and medium run-lengths, and it is comparable to all on-line versions in long ones. Considering only the on-line versions, all of them are equivalent to the naive one for runs of 10 and 100 seconds.

Full MA. In Figure 3(a), we depict the results achieved by the seven versions of the full algorithm on the homogeneous instances. The off-line version always appears to be the best choice. The default version achieves very good results, too. The difference in the performance of the off-line and the on-line versions decreases as the CPU time increases. The results achieved by the on-line versions are very similar to each other.

The results obtained on the heterogeneous instances, reported in Figure 3(b), suggest similar conclusions. In particular, the off-line version is the best performing for the short run-length, while this is not true for runs of 31 and 100 seconds. The relative performance of the off-line and the on-line versions follows the trend identified for the homogeneous instances: as the CPU time grows, the on-line versions achieve relatively better performance. This trend is even more evident here, since the off-line version is comparable to all the on-line ones for runs of 100 seconds. The results achieved by the random version are quite poor, while the naive version is always comparable to at least an on-line one.

Summary of the results. By examining the results just presented, we cannot identify a clear relation between the quality of the algorithms and the relative performance of off-line and on-line tuning: off-line tuning achieves quite

constantly very good performance, and thus it appears the most advantageous and conservative choice. Nonetheless, when a low quality algorithm is to be used, applying an on-line method may be preferable for short run-lengths. One critical issue in this case is the selection of the best on-line tuning method: on one hand, adaptive pursuit achieves always quite good results; on the other hand, in several cases it is not the best performing method. A further element that cannot be neglected is the relatively good performance achieved by the naive version, compared to the more advanced methods proposed in the literature. Still, by counting the cases in which each operator is winning against the others, we can conclude that adaptive pursuit is the on-line method to choose: it achieves in general good performance, and it often outperforms the naive version. Nonetheless, if we consider the effort devoted by the scientific community to the development and the analysis of on-line tuning methods, the difference between them and the naive version is surprisingly small.

Maybe surprisingly, the heterogeneity of the set of instances to be tackled does not have a remarkable impact on the results.

These conclusions are supported by further results we have obtained by performing the same analysis on two ant colony optimization (ACO) algorithms, namely $\mathcal{MAX}$–$\mathcal{MIN}$ ant system ($\mathcal{MM}$AS) for the QAP either with or without local search. We applied the on-line tuning methods described by Pellegrini et al. [11]. We tuned parameter α, that is, the exponent value used for the pheromone trails in the state transition rule. The results of this analysis are depicted in a supplementary report [25].

6 Conclusions

In this paper, we studied the performance of three memetic algorithms for the QAP, when their configurations are tuned either off-line or on-line. We consider one off-line and three on-line methods, we tested the algorithms on two different instance sets, a heterogeneous and a homogeneous one, and we observed the impact of the different tuning methods as a function of the quality of the algorithm.

The results do not allow drawing any clear conclusion on the relation between the tuning methods and the quality of the algorithms. In general, off-line tuning seems to be preferable under all experimental conditions. The heterogeneity of the instances to be tackled does not have a remarkable impact on the results. Some trend can be detected that indicates that, for a low quality algorithm, on-line tuning may achieve better results than off-line tuning. In this case, the choice of the on-line method to implement is not trivial and it must be done after considering the computational time available.

In future studies, we will try to further investigate the relation between the quality of the algorithms and the impact of off-line and on-line tuning. An extensive experimental analysis will be necessary to this aim. Moreover, we will increase the heterogeneity of the instances to be tackled, for identifying whether and for what level of heterogeneity on-line methods have a clear advantage over

off-line ones. Finally, we will further focus on the relative performance of the state-of-the-art on-line tuning methods compared to some simple approaches for perturbing the configuration used during the search process. Recently, Fialho [10] has proposed a well performing on-line method called rank-based multi-armed bandit. We will implement this further method and analyze its performance in our setting. In this framework, it may be interesting to identify some conditions under which the additional effort required for selecting a specific on-line method and for implementing it, is or is not payed in terms of improved performance.

Acknowledgments. This research has been supported by "META-X", an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium, and by "E-SWARM – Engineering Swarm Intelligence Systems", an European Research Council Advanced Grant awarded to Marco Dorigo (Grant Number 246939). The work of Paola Pellegrini is funded by a Bourse d'excellence Wallonie-Bruxelles International. Mauro Birattari and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS, of which they are Research Associates.

References

1. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Langdon, W., et al. (eds.) GECCO 2002, pp. 11–18. Morgan Kaufmann Publishers, San Francisco (2002)
2. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein, T., Blesa Aguilera, M.J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., Sampels, M. (eds.) HM 2007. LNCS, vol. 4771, pp. 108–122. Springer, Heidelberg (2007)
3. Adenso-Díaz, B., Laguna, M.: Fine-tuning of algorithms using fractional experimental designs and local search. Operations Research 54(1), 99–114 (2006)
4. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. J. Artif. Intell. Res. (JAIR) 36, 267–306 (2009)
5. Battiti, R., Brunato, M., Mascia, F.: Reactive Search and Intelligent Optimization. Operations Research/Computer Science Interfaces, vol. 45. Springer, Berlin (2008)
6. 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)
7. Maturana, J., Fialho, A., Saubion, F., Schoenauer, M., Sebag, M.: Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In: IEEE Congress on Evolutionary Computation, pp. 365–372 (2009)
8. Eiben, A.E., Michalewicz, Z., Schoenauer, M., Smith, J.E.: Parameter control in evolutionary algorithms. In: Lobo, F., Lima, C.F., Michalewicz, Z. (eds.) Parameter Setting in Evolutionary Algorithms, pp. 19–46. Springer, Berlin (2007)
9. Whitacre, J.M., Pham, Q.T., Sarker, R.A.: Credit assignment in adaptive evolutionary algorithms. In: Cattolico (ed.) GECCO 2006, pp. 1353–1360. ACM, New York (2006)
10. Fialho, A.: Adaptive Operator Selection for Optimization. PhD thesis, Université Paris-Sud XI, Orsay, France (2010)

11. Pellegrini, P., Stützle, T., Birattari, M.: Off-line and on-line tuning: a study on MAX-MIN ant system for TSP. In: Dorigo, M., Birattari, M., Di Caro, G.A., Doursat, R., Engelbrecht, A.P., Floreano, D., Gambardella, L.M., Groß, R., Şahin, E., Sayama, H., Stützle, T. (eds.) ANTS 2010. LNCS, vol. 6234, pp. 239–250. Springer, Heidelberg (2010)
12. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Technical Report Caltech Concurrent Computation Program, 826, California Institute of Technology, Pasadena, CA, USA, 1989.
13. Merz, P., Freisleben, B.: Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. IEEE Transactions on Evolutionary Computation 4(4), 337–352 (2000)
14. Merz, P., Freisleben, B.: A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem. In: Proc. Congress on Evolutionary Computation, pp. 2063–2070. IEEE Press, Los Alamitos (1999)
15. Merz, P., Freisleben, B.: A genetic local search approach to the quadratic assignment problem. In: 7th International Conference on Genetic Algorithms, pp. 465–472. Morgan Kaufmann, San Francisco (1997)
16. Goldberg, D.E.: Genetic algorithms and rule learning in dynamic system control. In: International Conference on Genetic Algorithms and Their Applications, pp. 8–15. Morgan Kaufmann Publishers Inc., San Francisco (1985)
17. Davis, L.: Applying adaptive algorithms to epistatic domains. In: Proc. of IJCAI, pp. 162–164 (1985)
18. Corne, D.W., Oates, M.J., Kell, D.B.: On fitness distributions and expected fitness gain of mutation rates in parallel evolutionary algorithms. In: Guervós, J.J.M., Adamidis, P.A., Beyer, H.-G., Fernández-Villacañas, J.-L., Schwefel, H.-P. (eds.) PPSN VII. LNCS, vol. 2439, pp. 132–141. Springer, Heidelberg (2002)
19. Thierens, D.: An adaptive pursuit strategy for allocating operator probabilities. In: IEEE Congress on Evolutionary Computation, pp. 1539–1546. IEEE Press, Piscataway (2005)
20. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning 47(2), 235–256 (2002)
21. Stützle, T., Hoos, H.H.: MAX–MIN ant system. Future Generation Computer Systems 16(8), 889–914 (2000)
22. Burkard, R., Karisch, S., Rendl, F.: QAPLIB – A quadratic assignment problem library. Journal of Global Optimization (10), 391–403 (1997)
23. Stützle, T., Fernandes, S.: New benchmark instances for the QAP and the experimental analysis of algorithms. In: Gottlieb, J., Raidl, G. (eds.) EvoCOP 2004. LNCS, vol. 3004, pp. 199–209. Springer, Heidelberg (2004)
24. Chiarandini, M.: Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems, ch. 3. PhD thesis, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany (2005)
25. Francesca, G., Pellegrini, P., Stützle, T., Birattari, M.: Companion to Off-line and On-line Tuning: a study on operator selection for a memetic algorithm applied to the QAP (2010), `http://iridia.ulb.ac.be/supp/IridiaSupp2010-015/`, IRIDIA Supplementary page.