Tuning Algorithms for Tackling Large Instances: An Experimental Protocol

Franco Mascia(✉), Mauro Birattari, and Thomas Stützle

IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
{fmascia,mbiro,stuetzle}@ulb.ac.be

Abstract. Tuning stochastic local search algorithms for tackling large instances is difficult due to the large amount of CPU-time that testing algorithm configurations requires on such large instances. We define an experimental protocol that allows tuning an algorithm on small tuning instances and extrapolating from the obtained configurations a parameter setting that is suited for tackling large instances. The key element of our experimental protocol is that both the algorithm parameters that need to be scaled to large instances and the stopping time that is employed for the tuning instances are treated as free parameters. The scaling law of parameter values, and the computation time limits on the small instances are then derived through the minimization of a loss function. As a proof of concept, we tune an iterated local search algorithm and a robust tabu search algorithm for the quadratic assignment problem.

Keywords: Automatic algorithm configuration · Scaling of parameters · Iterated local search · Robust tabu search · Quadratic assignment problem

1 Introduction

Many applications require the solution of very large problem instances. If such large instances are to be solved effectively, the algorithms need to operate at appropriate settings of their parameters. As one intriguing way of deriving appropriate algorithm parameters, the automatic configuration of algorithms has shown impressive advances [1]. However, tuning algorithms for very large instances directly is difficult, a main reason being the high computation times that even a single algorithm run on very large instances requires. There are two main reasons for these high computation times. First, the computational cost of a single search step scales with instance size; second, larger instances usually require a much larger number of search steps to find good quality solutions. From a theoretical side, the tuning time would scale linearly with the number of configurations tested or linearly with the computation time given to each instance. However, even if a limited number of algorithm configurations are tested during the tuning of the algorithm, the sheer amount of time required to test a single algorithm configuration on a very large instance makes it a problem of practical relevance also for the tuning.

Here, we define a protocol for tuning a stochastic local search (SLS) algorithm on small instances that allows us to make predictions on the behaviour of the tuned SLS algorithm on very large instances. To do so, we optimise the value of free variables of the experimental setting that allow us to make this kind of extrapolations. To give a concrete example, in this paper we present a case study on an iterated local search (ILS) [2,3] algorithm and a robust tabu search (RoTS) [4] algorithm for the quadratic assignment problem (QAP). In the ILS case, we intend to study the strength of the perturbation while in the case of RoTS, we intend to define the appropriate tabu list setting (or better said, finding an appropriate range for the tabu list length settings). Hence, we need to identify the scaling law for these two variables in the respective algorithms to very large instances.

For this we define an experimental protocol where we actually allow two free variables in our experimental setting: a policy for the cut-off time and a policy for the actual parameter configuration. The rationale of having also the cut-off time as a free variable (in addition to the actual parameter setting) is to find an appropriate cut-off time for training on small instances that allows us to extrapolate the parameter configuration for a target cut-off time on a very large instance. As an illustrative example, consider an ILS algorithm that exposes a single parameter to the tuning, for example, the one that controls the amount of perturbation of the current solution. This parameter acts on the balance between intensification and diversification of the algorithm. A reasonable assumption is that the amount of intensification and diversification determined by the parameter value depends on the instance size, and more specifically, that on very large instances the amount of diversification required is smaller due to the fact that the search space to be explored is already large and the algorithm will have to spend most of the time intensifying. In such cases a small cut-off time when tuning on small instances, can lead to algorithm configurations that imply stronger intensification and that therefore allow for a more realistic prediction for very large instances.

The structure of the paper is as follows. Section 2 presents the formalisation of the experimental setting. Section 3 presents a proof of concept with an ILS algorithm and a RoTS algorithm for the QAP. In Sect. 4 we draw the conclusions.

2 Modelling

In the most general case, we want to define a protocol for tuning an SLS algorithm on a set of small training instances $s \in S$ and make predictions on the behaviour of the tuned algorithm on a very large instance $s^\star$. There are two free variables in our experimental setting. The first one is the maximum cut-off time on the small instances, which we define as a policy $t(s)$ that depends on the instance size s. The second one is the parameter setting that we define as the policy $\hat{m}(s; t(s))$ that depends on the instance size and the cut-off time $t(s)$.

Our aim is to optimise policies $t(s)$ and $\hat{m}(s; t(s))$ to predict a good parameter setting $\hat{m}(s^\star; T^\star)$ when executing the algorithm on a very large instance $s^\star$ with cut-off time $T^\star$, where $T^\star$ is the maximum time-budget available for solving the target instance $s^\star$.

We cast this problem as a parameter estimation for the minimisation of a loss function. More in detail, we select a priori a parametric family for the policies $t(s)$ and $\hat{m}(s; t(s))$. The value defined by the policies for an instance size s will be determined by the sets of parameters π_t and $\pi_{\hat{m}}$. The number and type of parameters in π_t and $\pi_{\hat{m}}$ depend on the parametric family chosen for the respective policies. We further constrain the policies by requiring that the maximum cut-off time is larger than a specified threshold $t(s) > \delta$ and that the policy defines a specific cut-off time for the target instance $t(s^\star) = T^\star$.

Very small instances should have a smaller impact on the optimisation of the policies than have small or small-to-medium training instances. The latter are in fact closer and more similar to the target instance size $s^\star$. Therefore, in the most general case, we also use a weighting policy $\omega(s)$ of a specific parametric family with parameters π_ω. The only constraint on this policy is that $\sum_{s \in S} \omega(s) = 1$.

We define the loss function in Eq. 1 as the difference between $C_{\hat{m}}(s; t(s))$, which is the cost obtained when executing the algorithm with the parameter setting determined by $\hat{m}(s; t(s))$; and $C_B(s; t(s))$, which is the cost function obtained when executing the algorithm with the best possible parameter setting $B(s; t(s))$ given the same maximum run-time $t(s)$, and try to determine:

$$\underset{\pi_\omega, \pi_{\hat{m}}, \pi_t}{\arg\min} \sum_{s \in S} \omega(s) \left[C_{\hat{m}}(s; t(s)) - C_B(s; t(s)) \right]. \tag{1}$$

By finding the optimal settings for π_ω, $\pi_{\hat{m}}$, and π_t, we effectively find the best scaling of the examples in S, and the best cut-off time, which allow us to find the policy that best describes how the parameter setting scales with the sizes in S. The same policy can be used to extrapolate a parameter setting for a target instance size $s^\star$ and a target cut-off time $T^\star$.

3 A Proof of Concept

In this paper, we present a proof of concept in which we concretely use the parameter estimation in Eq. 1 to tune an ILS algorithm and a RoTS algorithm for the QAP [5].

The QAP models the problem of finding a minimal cost assignment between a set P of facilities and a set L of locations. Between each pair of facilities there is a flow defined by a flow function $w : P \times P \to \mathbb{R}$, and locations are at distance $d : L \times L \to \mathbb{R}$. To simplify the notation, flow and distance functions can be seen as two real-valued square matrices W and D respectively. The QAP is to find a bijective function $\pi : P \in L$ that assigns each facility to a location and that minimises the cost functional:

$$\sum_{i,j \in P} w_{i,j} d_{\pi(i),\pi(j)}.$$

3.1 Iterated Local Search

In our ILS algorithm for the QAP [3], after generating a random initial solution, a first improvement local search is applied until a local minimum is reached. Then the algorithm undergoes a series of iterations until the maximum cut-off time is reached. At each iteration the current solution is perturbed by a random k-exchange move. After the perturbation, an iterative improvement algorithm is applied until a local optimum is reached. The new solution obtained is accepted if and only if it improves over the current solution. A parameter k specifies the size of the perturbation. It is the only parameter exposed to the tuning, and it can assume values from 2 up to the instance size.

The Experimental Setting. For each size in $S = \{40, 50, \dots, 100\}$, we generate 10 random instances of Taillard's structured asymmetric family [6]. We then measure the average percentage deviation from the best-known solution for each size $s \in S$, by running the ILS algorithm 10 times on each instance with 100 values of the perturbation parameter and by taking the mean value. The maximum cut-off time for these experiments is set to a number of CPU-seconds that is larger than a threshold $max_t(s)$, which allows at least for 1000 iterations of the ILS algorithm with the perturbation strength set to $k = 0.5s$.

We fix the scaling policy as $\omega(s) = \frac{s^3}{\sum_{s\in S} s^3}$ with no parameters π_ω to be optimised. The parametric family of the parameter policy is the linear function $\hat{m}(s; t(s)) = c + ms$ with the parameters $\pi_{\hat{m}} = (c, m)$. The cut-off time policy $t(s)$ is defined as a function $t(s) = c_0 + c_1 s^\alpha$, with the constraint that $t(s) > \delta\, \forall s > s'$, where s' is the smallest $s \in S$. The constant δ is set to 20 ms as the minimum amount of time that can be prescribed for an experiment. Moreover, since we pre-computed the cost function for a given maximum cut-off time, we set also an upper-bound $t(s) < 3\ max_t(s)$. Finally, the policy has to pass through the point $(s^\star, T^\star)$, hence one of the parameters can be determined as function of the others and π_t can be restricted to the two parameters (c_0, α):

$$\underset{c,m,c_0,\alpha}{\arg\min} \sum_{s\in S} \frac{s^3}{\sum_{s\in S} s^3} \left[C_{\hat{m}}(s; t(s)) - C_B(s; t(s)) \right]. \tag{2}$$

To minimize the loss in Eq. 2 we implemented, for this case study, an ad hoc local search procedure that estimates the parameter values within predefined ranges. The local search starts by generating random solutions until a feasible one is obtained. This initial random solution is then minimised until a local optimum is reached. The algorithm is repeated until the loss function is equal to zero or a maximum number of iterations is reached. To minimise the incumbent solution, the algorithm selects an improving neighbour by systematically selecting a parameter and increasing or decreasing its value by a step l. For integer-valued parameters l is always equal to 1. For real-valued parameters the value of l is set as in a variable neighbourhood search (VNS) [10,11]. Initially, l is set to 1^{-6}, then as soon as the the local search is stuck in a local minimum, its value l is first increased to 1^{-5}, then 1^{-4} and so on until l is equal to 1. As soon as the local search escapes the local minimum, the value of l is reset to 1^{-6}. With this local search, we do not want to define an effective procedure for the parameter estimation, the aim here is mainly to have some improvement algorithm for finding reasonable parameter settings for our policy and to present a proof of concept of the experimental protocol presented in this paper. In the future, we plan to replace our ad hoc local search with more performing local search algorithms for continuous optimization such as CMA- ES [7] or Mtsls1 [8]. The parameters c and m are evaluated in the range $[-0.5, 0.5]$, and the parameter α in the range $[3, 4, \dots, 7]$. The parameter c_0 is initialised to $max_t(s')$, where s' is the smallest $s \in S$, and evaluated in the range $[0, 3\, max_t(s')]$.

Results. For each target size $s^\star$ and target cut-off time $T^\star$, we first let the $t(s)$ policy pass through the point $(s^\star, T^\star)$, and then we find the optimal policies by minimising Eq. 2. We optimise and test our policies for the target sizes $s^\star$ 150, 200, 300, 400 and 500. For the minimisation of the loss function, we allow our ad hoc local search algorithm to restart at most 25 times.

To evaluate the policies obtained, we compute two metrics: the loss obtained by the predicted value and a normalised score inspired by [9]. The loss is the difference between the cost obtained when running the algorithm with the parameter prescribed by the policy and the best parameter for the given size and cut-off time. The normalised score, is computed as:

$$\frac{E_{unif}C(s;t(s)) - C_{\hat{m}}(s;t(s))}{E_{unif}C(s;t(s)) - C_B(s;t(s))},$$

where $E_{unif}C(s;t(s))$ is the expectation of an uniform choice of the parameter setting. This score is equal to zero when the cost of the parameter prescribed by the policy is the same as the one expected by an uniform random choice of the parameter. It is equal to one when the cost of the prescribed parameter corresponds to the cost attainable by an oracle that selects the best parameter setting. Negative values of the score indicate that the prescribed parameter is worse than what could be expected by an uniform random choice.

To calculate the two metrics, we pre-compute on the target instance sizes, the values of the cost function for 100 possible parameter configurations. Then, to evaluate the predicted values, we round them to the closest pre-computed ones.

In Fig. 1 we present the results for the largest of the test instances, with $s^\star = 500$ and $T^\star = 6615.44$ CPU-seconds. The plot on top shows the cut-off time policy with exponent $\alpha = 4$ that passes through the target size and target cut-off time. The second plot from the top shows the linear policy for the parameter setting that prescribes a perturbation of size 177, while the optimal perturbation value for the specified cut-off time is 171. The third plot shows the loss. The predicted parameter is rounded to the closest precomputed one, which is 176. The difference in the average deviation from the best known solution amounts to 0.044977. The last plot at the bottom shows the normalised score that for the prediction on target size 500 is equal to 0.841609. In Table 1 we summarise similar results also for 150, 200, 300 and 400.

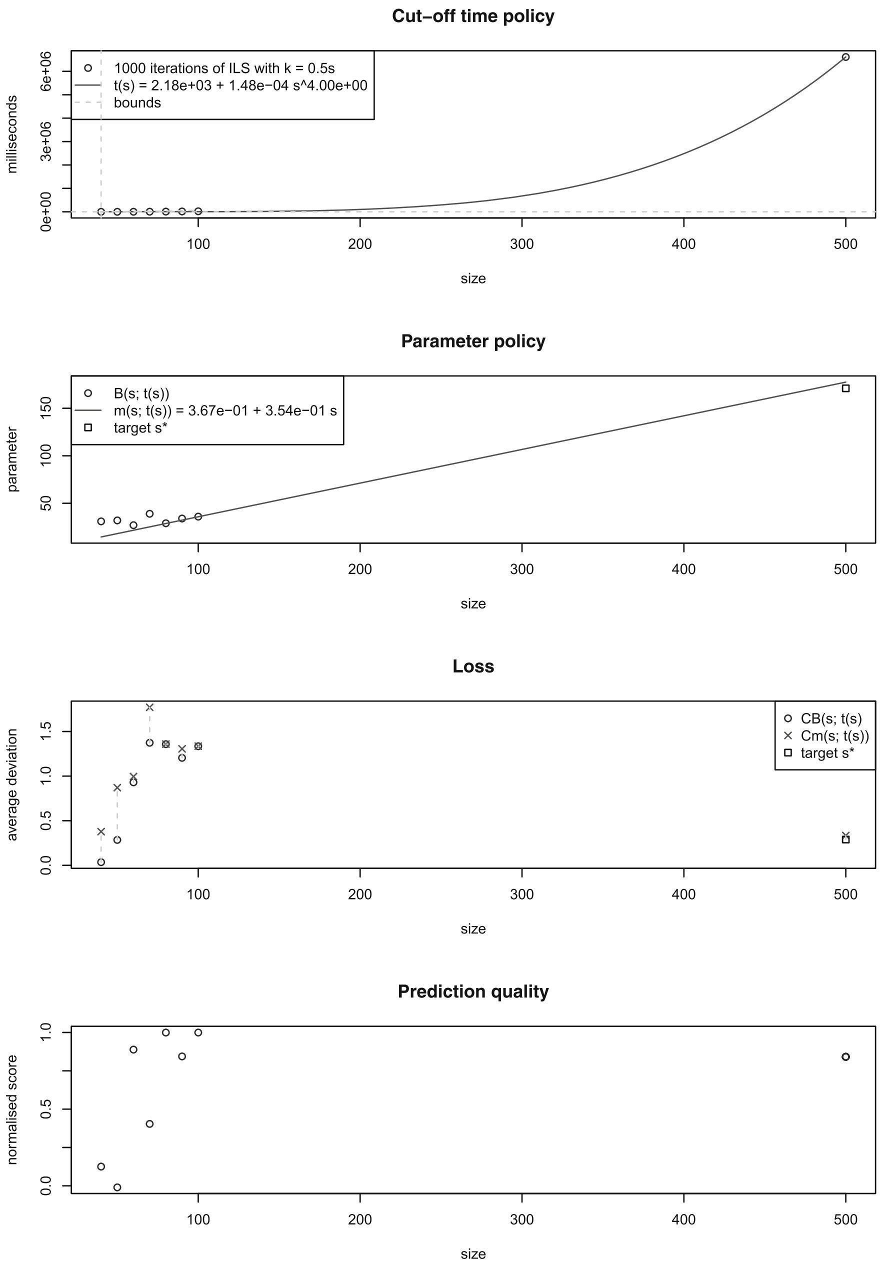

Fig. 1. Cut-off time policy, parameter policy for ILS, loss, and prediction quality on target instance size $s^\star = 500$.

Table 1. Summary of the loss and normalised score on the target sizes of the policies optimised for ILS.

$s^\star$	$T^\star$	Loss	Normalised score
150	102.85	0.058957	0.880454
200	257.50	0.084706	0.814831
300	1 047.82	0.039883	0.887496
400	2 596.33	0.039318	0.881627
500	6 615.44	0.044977	0.841609

To further evaluate the policies obtained, we also compare them with a dynamic setting of the parameter as in a VNS algorithm. This comparison is relevant, as a dynamic variation of the perturbation size as propagated in VNS would be a reasonable way of addressing the fact that a single best perturbation size value is unknown for the very large instances. For each target size $s^\star$ in our test set, we run both algorithms 10 times on the 10 instances of size $s^\star$. Figure 2 shows the average deviation of the two algorithms with respect to the results that are obtained with the a posteriori best parameter for the given size and cut-off time. Also in this case, the policies obtained for the parameter setting lead to results which are much better than what can be expected from a baseline VNS algorithm. A stratified rank-based permutation test (akin to a stratified version of the Mann-Whitney U test) rejects at a 0.05 significance level the null hypothesis of no difference between the average deviations obtained with the two algorithms.

To test for the importance of optimising also a policy for the cut-off time, we tested a tuning protocol in which the parameter policy is the only free variable being optimised. In this case we fixed the cut-off time to a number of CPU-seconds that allows for 1000 steps of the ILS algorithm on the target instance size $s^\star$ with a perturbation size $k = 0.5s$. As shown in Table 2, on all target instance there is a clear advantage of leaving the cut-off time policy as a free variable of the experimental setting.

Table 2. Normalised score on the target sizes for ILS in the case in which the cut-off time is optimised as a free variable of the experimental setting, and in the case in which the cut-off time is fixed.

$s^\star$	$T^\star$	Cut-off time policy	Fixed cut-off time
150	102.85	0.880454	0.742731
200	257.50	0.814831	0.769348
300	1 047.82	0.887496	0.661414
400	2 596.33	0.881627	0.606192
500	6 615.44	0.841609	0.701780

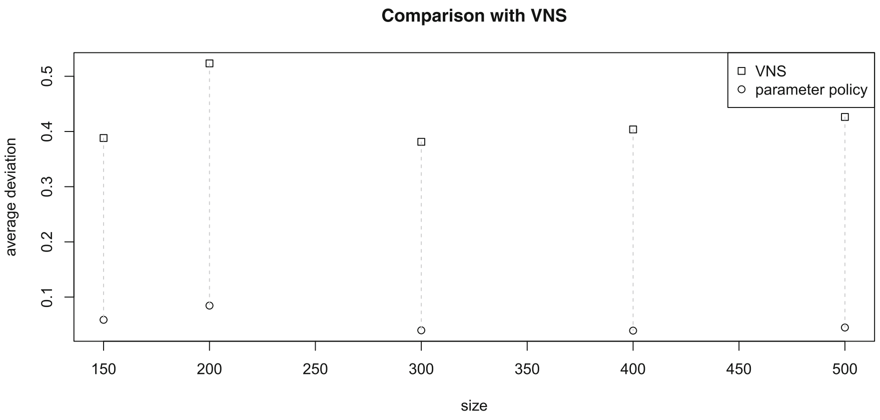

Fig. 2. Comparison between VNS and the parameter policy for ILS on target instances $s^\star$ at time $T^\star$.

3.2 Robust Tabu Search

The RoTS algorithm for the QAP [4] is a rather straightforward tabu search algorithm that is put on top of a best improvement algorithm making use of the usual 2-exchange neighbourhood for the QAP, where the location of two facilities are exchanged at each iteration. A move is tabu, if at least the two facilities involved are assigned to a location they were assigned in the last tl iterations, where tl is the tabu list length. Diversification is ensured by enforcing specific assignments of facilities to locations if such an assignment was not considered for a rather large number of local search moves. In addition, an aspiration criterion is used that overrides the tabu status of a move if it leads to a new best solution. The term *robust* in RoTS stems from the random variation of the tabu list length within a small interval; this mechanism was intended to increase the robustness of the algorithm by making it less dependent on one fixed setting of tl [4]. Hence, instead of having a fixed tabu list length, at each iteration the value for the tabu tenure is selected uniformly random in the range $\max(2, \text{unif}(\mu - 0.1, \mu + 0.1) \cdot s))$. Thus, μ is the expected value of the tabu list length and it is the only parameter exposed to the tuning. In the original paper, a setting of $\mu = 1.0$ was proposed.

The Experimental Setting. For instance sizes $S = \{40, 50, \ldots, 100\}$, we generate 10 Taillard's instances with uniformly random flows between facilities and uniformly random distances between locations [6]. For each parameter setting of $\mu \in \{0.0, 0.1, \ldots, 2.5\}$, and for each instance size, we compute the mean deviation from the best-known solution. The mean is computed over 10 runs of the RoTS algorithm on each of the 10 instances. The maximum cut-off time for these experiments is set to a number of CPU-seconds that allow for at least $100 \cdot s$ iterations of the RoTS algorithm.

We keep for this problem the same free variables and the same parametric families we used for the ILS algorithm. The only difference is a further constraint on the parameter policy $\hat{m}(s; t(s))$ that is required to prescribe a positive value for the parameter for all $s \in S$ and on the target size $s^\star$. Since the problem is more constrained and harder to minimise, we allow our ad hoc local search algorithm to restart from at most 5000 random solutions. We optimise and test the policies on target instance sizes 150, 200, 300, 400 and 500.

Results. As for the ILS algorithm, we evaluate the policies by measuring the loss on the target instance sizes and by computing the normalised score. In Fig. 3 we present the results on the largest test instance $s^\star = 500$. On this instance, the parameter policy prescribes a parameter setting of 0.040394 while the best parameter for this instance size and cut-off time is 0.1. The loss amounts to 0.051162 and the normalised score is 0.653224.

Table 3 summarises the results on all test instances. On instance $s^\star = 400$ the parameter policy obtains a loss of 0 and a normalised score equal to 1. This is due to the fact that for evaluating the policies, and hence knowing the best parameter setting given the size and cut-off time, we pre-computed the cost of a fixed number of parameter configurations. When computing the cost of the parameter configuration prescribed by the policy, the value of the parameter is rounded to the closest value for which the cost has been pre-computed. For instance, for size $s^\star = 400$ the prescribed parameter value is 0.086961. This value is rounded to 0.1, which corresponds to the best parameter setting.

To further evaluate the policy we evaluate it with a default setting of the parameter that, in the case of RoTS, would be $\mu = 1.0$. Figure 4 shows the average deviation obtained with the parameter setting prescribed by the policy and the default parameter setting. The average deviation is computed with respect to the solution quality obtained when using the best a posteriori parameter setting given the instance size and the cut-off time. On all instances the policy obtained for the parameter setting lead to results which are much better than what we could expect from the default parameter setting. Also in this case, a stratified rank-based permutation test rejects at a 0.05 significance level the null hypothesis of no difference between the average deviations obtained with the two algorithms.

Table 3. Summary of the loss and normalised score on the target sizes of the policies optimised for RoTS.

$s^\star$	$T^\star$	Loss	Normalised score
150	120.09	0.038592	0.827993
200	278.84	0.062164	0.697857
300	954.89	0.021499	0.861288
400	2 517.97	0	1
500	5 473.06	0.051162	0.653224

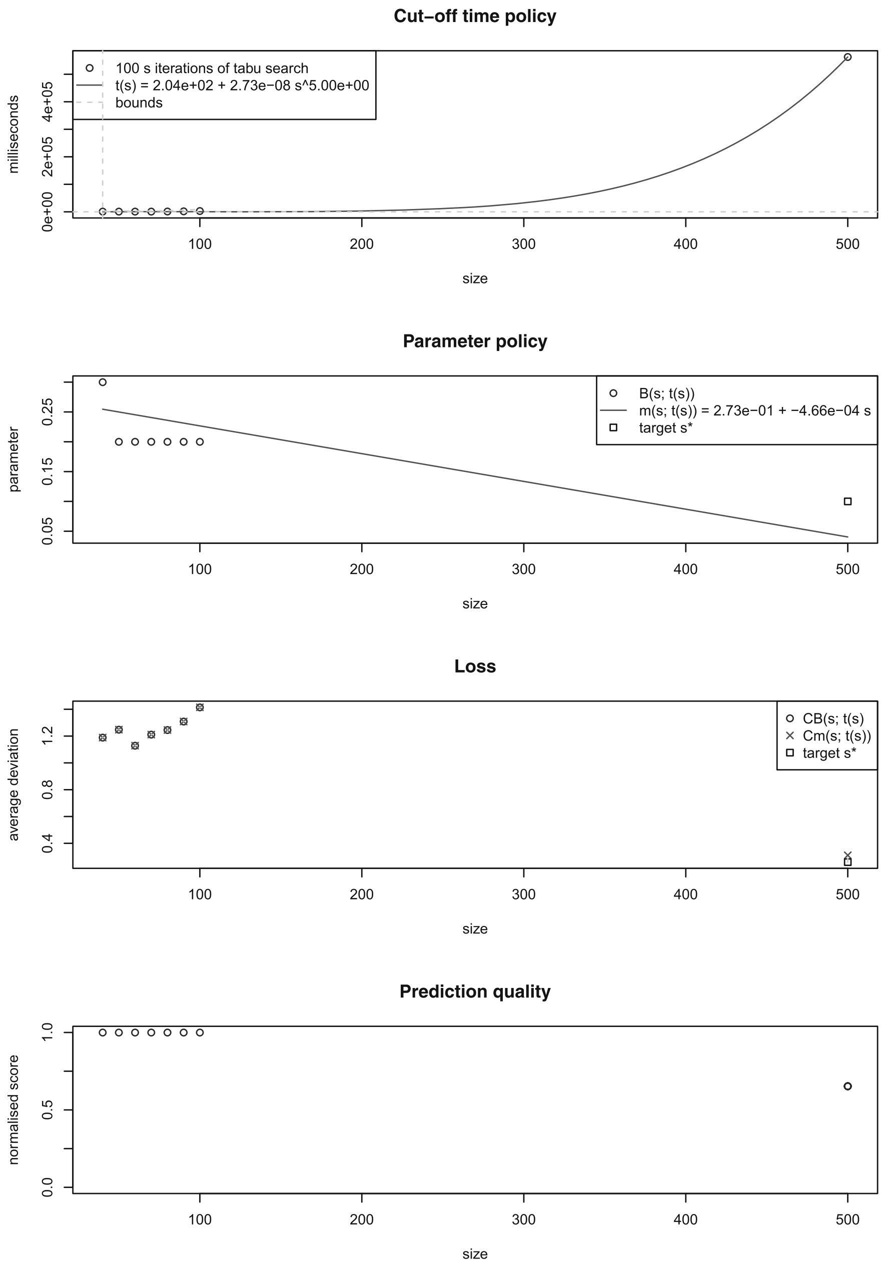

Fig. 3. Cut-off time policy, parameter policy for RoTS, loss, and prediction quality on target instance size $s^\star = 500$.

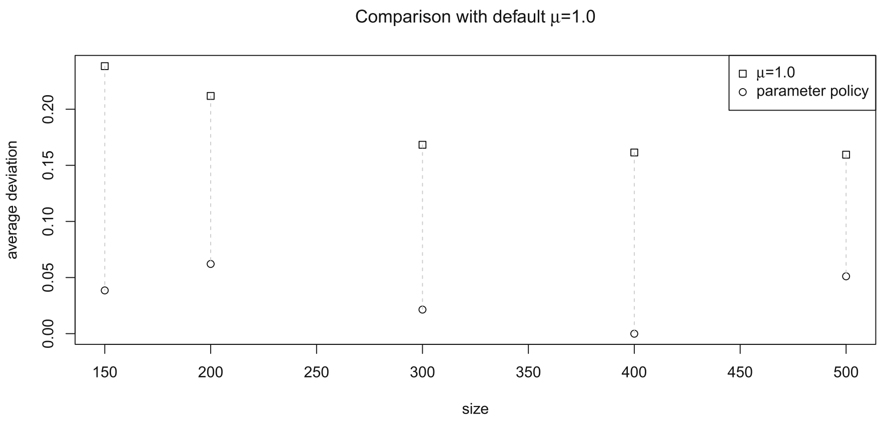

Fig. 4. Comparison between the default value $\mu = 1.0$ and the parameter policy for RoTS on target instances $s^\star$ at time $T^\star$.

To test the importance of optimising also a policy for the cut-off time, we tested also for this problem a tuning protocol in which we optimise the parameter policy while keeping the cut-off time fixed. In this case the cut-off time was fixed to a number of CPU-seconds that allow for $100 \cdot s$ steps of the RoTS algorithm. On this problem there was no clear-cut result, in fact on instance sizes 300, 400, and 500, with a fixed cut-off time policy the score remains the same; on instance 150 the score drops from 0.827993 to 0.354483; and on instance size 200 the score increases from 0.697857 to 1.

4 Conclusions and Future Work

We presented an experimental protocol in which optimising the value of the free variables of the experimental setting allows for tuning an SLS algorithm on a set of small instances and extrapolating the results obtained on the parameter configuration for tackling very large instances. We cast the problem of optimising the value of the free variables as a parameter estimation for the minimisation of a loss function. In the general formulation as well as in the proofs of concept presented in this paper, we suggested as possible free variables: (i) a policy for scaling the parameter configuration, (ii) a policy for selecting the cut-off time when tuning the algorithm on the small instances, and (iii) a policy for weighting the small instances during the minimisation of the loss function.

We presented a study on an ILS algorithm and a RoTS algorithm with one parameter for the QAP. On both problems we obtained promising results, with the extrapolated parameter setting being close to best a posteriori parameter setting for the instances being tackled. We also showed that results obtained by our method are much better than default static or dynamic settings of the parameters. We believe that our approach may be a viable way of tuning algorithms for very large instances if SLS algorithms rely on few key parameters such as the algorithms tested here.

One key element of our contribution is the optimisation of a policy for the cut-off time that prescribes how long a configuration should be tested during the tuning on small instances. We showed experimentally, that at least for the ILS algorithm, optimising a cut-off time policy allows for better extrapolations of the parameter setting on large instances.

As future work, we plan to extend the approach to algorithms with (many) more than one parameter and to extrapolate to much larger instance sizes. In both cases we also need to define an extended protocol for assessing the performance of our method since pre-computing the cost function may become prohibitive. Furthermore, an automatic selection of the parametric models, and a comparisons to other recent approaches for tuning for large instances such as [12] would be interesting.

Acknowledgments. This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Franco Mascia, Mauro Birattari, and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS. The authors also acknowledge support from the FRFC project "*Méthodes de recherche hybrids pour la résolution de problèmes complexes*". This research and its results have also received funding from the COMEX project within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office.

References

1. Hoos, H.H.: Programming by optimization. Commun. ACM **55**(2), 70–80 (2012)
2. Lourenço, H.R., Martin, O., Stützle, T.: Iterated local search: framework and applications. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management Science, 2nd edn, pp. 363–397. Springer, New York (2010)
3. Stützle, T.: Iterated local search for the quadratic assignment problem. Eur. J. Oper. Res. **174**(3), 1519–1539 (2006)
4. Taillard, É.D.: Robust taboo search for the quadratic assignment problem. Parallel Comput. **17**(4–5), 443–455 (1991)
5. Koopmans, T.C., Beckmann, M.J.: Assignment problems and the location of economic activities. Econometrica **25**, 53–76 (1957)
6. Taillard, E.D.: Comparison of iterative searches for the quadratic assignment problem. Location Sci. **3**(2), 87–105 (1995)
7. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (2001)
8. Tseng, L.Y., Chen, C.: Multiple trajectory search for large scale global optimization. In: Proceedings of IEEE Congress on Evolutionary Computation, Piscataway, NJ, IEEE, pp. 3052–3059 June 2008
9. Birattari, M., Zlochin, M., Dorigo, M.: Towards a theory of practice in metaheuristics design: a machine learning perspective. Theor. Inform. Appl. **40**(2), 353–369 (2006)

10. Mladenovic, N., Hansen, P.: Variable neighbourhood search. Comput. Oper. Res. **24**(11), 71–86 (1997)
11. Hansen, P., Mladenovic, N.: Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**(3), 449–467 (2001)
12. Styles, J., Hoos, H.H., Müller, M.: Automatically configuring algorithms for scaling performance. In: Hamadi, Y., Schoenauer, M. (eds.) LION 6. LNCS, vol. 7219, pp. 205–219. Springer, Heidelberg (2012)