An Empirical Study of Off-line Configuration and On-line Adaptation in Operator Selection

Zhi Yuan[1⋆], Stephanus Daniel Handoko[2], Duc Thien Nguyen[2], and Hoong Chuin Lau[2]

[1] Professorship of Applied Mathematics, Department of Mechanical Engineering, Helmut-Schmidt-University, Hamburg, Germany
yuanz@hsu-hh.de

[2] School of Information Systems, Singapore Management University, Singapore
{dhandoko,dtnguyen,hclau}@smu.edu.sg

Abstract. Automating the process of finding good parameter settings is important in the design of high-performing algorithms. These automatic processes can generally be categorized into off-line and on-line methods. Off-line configuration consists in learning and selecting the best setting in a training phase, and usually fixes it while solving an instance. On-line adaptation methods on the contrary vary the parameter setting adaptively during each algorithm run. In this work, we provide an empirical study of both approaches on the operator selection problem, explore the possibility of varying parameter value by a non-adaptive distribution tuned off-line, and incorporate the off-line with on-line approaches. In particular, using an off-line tuned distribution to vary parameter values at runtime appears to be a promising idea for automatic configuration.

1 Introduction

The performance of metaheuristics in solving hard problems usually depends on their parameter settings. This leaves every algorithm designer and user with a question: how to properly set algorithm parameters? In recent years, many works on using automatic algorithm configuration to replace the conventional rule-of-thumb or trial-and-error approaches have been proposed [1–3].

The automatic algorithm configuration methods can generally be categorized into two classes: off-line method and on-line method. The goal of off-line configuration method, also referred to as parameter tuning, is to find a good parameter configuration for the target algorithm based on a set of available training instances [4]. These training instances in practice can be obtained from, e.g., a simulated instance generator or historical data if the target optimization problem happens in a recurring manner, for example, to optimize logistic plans on weekly delivery demand, etc. Once the training phase is finished, the target algorithm is deployed using the tuned configuration to solve future instances. The

⋆ Main part of this research was carried out while Zhi Yuan was working at the School of Information Systems, Singapore Management University. Zhi Yuan is currently also a PhD candidate at IRIDIA, CoDE, Université Libre de Bruxelles, Belgium.

off-line tuned configuration deployed is usually fixed when solving each instance, and across different instances encountered[3]. Existing approaches to this aim include, e.g., [6–11]. In contrast with off-line configuration, instead of keeping a static parameter configuration during the algorithm run, an on-line configuration method tries to vary the parameter value as the target algorithm is deployed to solve an instance. Such approaches are also referred to as parameter adaptation [12] or parameter control [13]. The on-line parameter adaptation problem has attracted many attentions and research efforts, especially in the field of evolutionary algorithm [14]. The usage of machine learning techniques in parameter adaptation is also the unifying research theme of reactive search [3].

Although the on-line and off-line methods approach the automatic algorithm configuration problem differently, they can be regarded as complementary to each other. For example, the on-line methods usually also have a number of hyper-parameter to be configured, and this can be fine-tuned by an off-line method, as done in the design of on-line operator selection method in [15]. Besides, off-line methods can provide a good starting parameter configuration, which is then further adapted by an on-line mechanism once the instances to be solved are given. [16] provides an in-depth analysis in this direction under the context of ant colony optimization algorithms (ACO), but shows that the on-line methods usually worsen the algorithm performance comparing with using a fixed configuration tuned off-line. Francesca et al. [17] compared on-line adaptation in operator selection with a static operator tuned off-line, and found using a statically tuned operator more preferable in their context. Another study in [18] shows that instead of adapting the tabu list length on-line as in reactive tabu search [19], varying the tabu list length by a static distribution tuned off-line performs better.

In this empirical study, we continue with [17] on the operator selection problem, and try to challenge the off-line tuned static operator by: 1) varying the parameter value by a non-adaptive off-line tuned distribution; 2) using off-line configuration in the design of on-line adaptive approaches; 3) cooperation of the non-adaptive approaches and the adaptive approaches. We also provide further analysis on the performance of on-line adaptation mechanisms.

2 The Target Problem and Algorithm

The target problem to be tackled is the quadratic assignment problem (QAP) [20]. In the QAP, n facilities are assigned to n locations, where the flow f_{ij} between each pair of facilities $i, j = 1, \ldots, n$ and the distance d_{ij} between each pair of locations $i, j = 1, \ldots, n$ are given. The goal is to find an permutation π that assigns to the location i one unique facility π_i, such that the cost defined

[3] There exist off-line configuration approaches called portfolio-based algorithm selection[5], which returns a portfolio of configurations instead of one fixed configuration, then select a configuration from the portfolio based on the feature of the future instance. However, each of its configurations remains fixed when solving an instance.

as the sum of the distances multiplied by the corresponding flows such as follows:

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

is minimized.

As the target algorithm for the study of off-line and on-line configuration methods, we focus on the operator selection in the evolutionary algorithm (EA). Our implementation of EA is inspired by the work described by Merz et al. [21]. In EA, a population of p individuals, each of which represents a QAP solution, are evolved from iteration to iteration by applying variation operators such as crossover and mutation. Initially, the p individuals are generated uniformly at random. Then at each iteration, p_c new individuals, dubbed offspring, will be generated by applying a crossover operator, and p_m new individuals will be generated by applying a mutation operator. All these new individuals may be refined by applying an optional local search procedure. The best p individuals among the old and newly generated individuals will be selected to enter the next iteration.

A crossover operator generates an offspring based on two through recombination of the chromosomes of two randomly chosen parent solutions. In this study, we look into the following four different crossover operators:

Cycle crossover (CX) first passes down all chromosomes that are shared by both parents, I_{p_1} and I_{p_2}, to the offspring, I_o. The remaining chromosomes of the offspring are assigned starting from a random one, $I_o(j)$. CX first sets $I_o(j) = I_{p_1}(j)$. Then, denoting $I_{p_1}(j')$ as the chromosomes where $I_{p_1}(j') = I_{p_2}(j)$, CX sets $I_o(j') = I_{p_1}(j')$ and substitutes the index j with j'. This procedure is repeated until all chromosomes of I_o are assigned.

Distance-preserving crossover (DPX) generates an offspring that has the same distance from both parents. DPX simply passes down to I_o all the chromosomes that are shared by both I_{p_1} and I_{p_2}. Each of the remaining chromosomes, $I_o(j)$, is assigned randomly provided that $I_o(j)$ is a permutation and it is different from both $I_{p_1}(j)$ and $I_{p_2}(j)$ in some approximate sense.

Partially-mapped crossover (PMX) randomly draws two chromosome locations of I_o, namely j and j' where $j < j'$. PMX then sets $I_o(k) = I_{p_1}(k)$ for all k outside the range of $[j, j']$ and $I_o(k) = I_{p_2}(k)$ for all $j \leq k \leq j'$. If the offspring generated is not a valid permutation, then for each chromosome pair $I_o(k)$ and $I_o(z)$ where $I_o(k) = I_o(z)$ and $j \leq z \leq j'$, PMX sets $I_o(k) = I_{p_1}(z)$. This process is repeated until a valid permutation is obtained.

Order crossover (OX) randomly draws two chromosome locations of I_o, namely j and j' where $j < j'$. OX then sets $I_o(k) = I_{p_1}(k)$ for all $j \leq k \leq j'$ and assigns in the k-th unassigned chromosomes of I_o the k-th chromosomes of I_{p_2} that differs from any $I_o(z)$, $j \leq z \leq j'$.

3 Operator Selection Strategies

3.1 The Static Operator Strategy

The static operator strategy (SOS) refers to fixing one operator when solving an instance. Most EA follows this strategy, especially when an off-line configuration tool is available [17]. Then this amounts to setting a categorical parameter.

3.2 The Mixed Operator Strategy

In contrast with fixing one operator to use, the mixed operator strategy[4] (MOS) assigns a probability to each operator. This allows an operator to be selected at each iteration of the algorithm under a certain probability. This strategy is often designed with a uniform probability distribution for each possible operators in the literature [22, 17], and referred to as "naive". Of course, the probability of selecting each operator can be set in other ways than uniform, and can be regarded as a real-valued parameter.[5] These parameters can potentially be fine-tuned in the off-line training phase.

3.3 The Adaptive Operator Selection

Different from the two approaches above, adaptive operator selection strategy (AOS) try to adjust the parameter values while running the target algorithm for each instance. As an on-line method, it is able to adapt parameter values according to different instances and different search stages. The development of such on-line methods needs to address two issues: the reward function, or credit assignment [23], which concerns how to measure operator quality according to operator performance; and the adaptation mechanism that concerns which operator to use at each time step according to the performance measurement.

Reward Function Two reward functions were used in our work. Both versions make reference of the cost of the offspring c_o to the cost of the current best solution c_b and the better parent solution c_p. The first reward function is adopted from the study of [17], for an operator i that is used to generate a set $\mathcal{I}_i$ of offspring at the current iteration:

$$R_1^i = \frac{1}{|\mathcal{I}_i|} \sum_{o \in \mathcal{I}_i} \cdot \frac{c_b}{c_o} \max\{0, \frac{c_p - c_o}{c_p}\}. \tag{2}$$

A drawback in the reward function R_1 is that the relative improvement of the offspring over its better parent will bias the multiplicative reward value much

[4] The term is syntactically and semantically analogous to the term *mixed strategy* widely used in game theory.

[5] One can even regard the static operator strategy as a degenerate case of a mixed strategy, in which one operator is selected with probability 1, and each of the others with probability 0.

stronger than its relative performance to the current best solution. This may not be effective especially when the parents are drawn uniformly randomly. We modify (2) by making the reference to the parent solution and the current best solution to contribute the same magnitude to the reward function:

$$R_2^i = \frac{1}{|\mathcal{I}_i|} \sum_{o \in \mathcal{I}_i} \frac{c_b}{c_o} \cdot \frac{c_p}{c_o} \cdot sign(c_p - c_o), \tag{3}$$

where $sign(x)$ function returns 1 when $x > 0$, and returns 0 otherwise.

On-line Adaptation Mechanisms We considered the three on-line algorithm adaptation methods studied in [17] for the operator selection problem, namely, Probability Matching (PM) [24], Adaptive Pursuit (AP) [25] and Multi-Armed Bandit (MAB) [26]. These three on-line methods update the quality Q_i of each candidate operator i by the formula:

$$Q_i = Q_i + \alpha(R^i - Q_i) \tag{4}$$

where $0 \leq \alpha \leq 1$ is a parameter, and Q_i is by default initialized to 1 for each operator i. Using a PM mechanism, the probability of choosing an operator i is given by the following formula:

$$P_i = P_{min} + (1 - |I|P_{min}) \frac{Q_i}{\sum_{i' \in I} Q_{i'}}, \tag{5}$$

where I is the set of all possible operators. The lower threshold $0 \leq P_{min} \leq 1$ is a parameter to guarantee that every operator has a chance to show its impact. The second adaptation method AP differs from PM by using a different probability update formula than Equation 5:

$$P_i = \begin{cases} P_i + \beta(P_{max} - P_c), & \text{if } Q_i = \max_{i'} Q_{i'} \\ P_i + \beta(P_{min} - P_c), & \text{otherwise,} \end{cases} \tag{6}$$

where $0 \leq \beta \leq 1$ is a parameter, and $P_{max} = 1 - (|I| - 1)P_{min}$. Over time, the probability of choosing a promising operator converge to P_{max} while all others descend into P_{min}. The third adaptation method MAB selects an operator $\bar{i}$ deterministically by

$$\underset{i \in I}{\operatorname{argmax}} \left\{ \bar{R^i} + \gamma \left(\sqrt{\frac{2 \ln \sum_{i'} n_{i'}}{n_i}} \right) \right\}, \tag{7}$$

where $\bar{R^i}$ is the average reward computed since the beginning of the search and n_i is the number of times the crossover operator i is chosen.

4 Experimental Setup

All experiments are conducted on a computing node with 24-core Intel Xeon CPU X7542 at 2.67GHz sharing 128GB RAM. Each run uses single thread.

4.1 Instance Setup

Three classes of QAP instances are considered in our experiments: one heterogeneous and two homogeneous sets. For the heterogeneous set (`het`), we followed the experimental setup in [17]: 32 instances from QAPLIB [27] with size from 50 to 100.[6] For the homogeneous sets, we generated 32 relatively easy homogeneous instances (`hom-easy`) and 32 harder homogeneous instances (`hom-hard`) using instance generator described in [28]. The instances in `hom-easy` are uni-size 80, with Manhattan distance matrix and random (unstructured) flow matrix generated with the same distribution with 50% sparsity; while the `hom-hard` instances are uni-size 100 with zero sparsity. Both homogeneous instance sets are chosen with large size (80 and 100), so that the computational overhead of the on-line adaptation mechanisms can be ignored.[7] All three instance classes are divided in half, 16 instances for training and 16 others for testing. Each instance was run 10 times, resulting in 160 instance runs. Each of the 160 runs is assigned with a unique random seed. Note that during each run, different algorithms will use the same random seed. This is to reduce evaluation variance [29].

4.2 Target Algorithm Setup

In [17], three memetic algorithm (MA) schemes were used for experiments: simple MA with crossover only; intermediate MA with crossover and local search; and full MA with crossover, mutation, and local search. Three levels of computation time are considered, 10, 31, and 100 seconds. From our initial experiments, we found that local search is time-consuming. For an instance of size 100, one local search took about 1 second. The intermediate and full MA thus performed no crossover in 10 or 31 seconds, and only 1 or 2 crossover generations after 100 seconds.[8] With this observation, and also to better distinguish the performance difference of crossover operator selection strategies, we excluded the local search as well as mutation[9], and focused on the crossover operation in this study. In such case, the computation time chosen corresponds to around 9000, 30 000, 90 000 crossover generations, respectively. For the default parameters in our implemented MA, we followed exactly [17], setting population size $p = 40$, crossover population $p_c = p/2 = 20$. A restart is triggered when the average distance over all pairs of individuals in the population has dropped below 10 or the average

[6] There are in total 33 instances found in the QAPLIB with size from 50 to 100. We further exclude one of them, esc64a, which is too simple and each algorithm considered in this work will solve it to optimum. Then it results in a total number of 32 instances in the heterogeneous set.

[7] Comparing with the non-adaptive operator strategy (fixed or mixed strategy), the computational overhead of the on-line adaptation mechanisms in our implementation is around 1% on instances of size 100, and around 3% on instances of size 50.

[8] More sophisticated techniques such as *don't look bit* or neighborhood candidate list may speed up local search. However, the development of these techniques is out of the scope of this study.

[9] However, mutation will be used in restart when the population converges.

Table 1. The hyper-parameters of the on-line adaptive operator selection: their default values and their ranges for off-line configuration.

param. name	used in	default	range	comment
α	MP, AP	0.3	[0.0, 1.0]	adaptation rate
P_{min}	MP, AP	0.05	[0.0, 0.2]	minimum probability
β	AP	0.3	[0.0, 1.0]	learning rate
γ	MAB	1.0	[0.0, 5.0]	scaling factor

fitness of the population has remained unchanged for the past 30 generations. In such case, each individual except the current best one will be changed by a mutation operator until it is 30% of the instance size differ from itself.

4.3 Off-line Configuration Setup

Configuring SOS The task is to choose one of the four crossover operators based on the training set. Since the parameter space is small, we assess each static operator by an exhaustive evaluation in each of the training set, which consists of 10 runs of 16 instances.

Configuring MOS Three versions of MOS are presented in this work: an untrained MOS with uniform probability distribution for each operator, denoted MOS-u and two automatically tuned versions of MOS, denoted MOS-b and MOS-w. The two tuned versions differ in how the configuration experiment is designed, more specifically, in which reference operator to choose: MOS-b chooses the best operator as reference, while MOS-w chooses the worst. Note that finding the best or the worst operator requires a priori knowledge such as studied in Section 5.1, or additional tuning effort. However, this additional tuning effort is usually small, since the parameter space is much smaller comparing with the rest tuning task. Suppose there are n operators, each of which is assigned a parameter $q_i, i = 1, \ldots, n$. After a reference operator r is chosen, in our case, either the best or worst operator, we fix $q_r = 1$, and try to tune the $n - 1$ parameters $q_i, i = \{1, \ldots, n\} \setminus \{r\}$. The range of these $n - 1$ parameters is set to $[0, 1.2]$ in MOS-b, while in MOS-w, the range is set to $[0, 100]$. Since the parameter space is infinite, exhaustive evaluation won't be feasible, thus we used two state-of-the-art automatic configuration tool, namely iterated racing [7] and BOBYQA post-selection [11, 30]. For each of the configuration methods, maximum 1000 target algorithm runs were allowed as configuration budget. Then the best configurations found by the both configurators are compared based on their training performance, and the one with the better training performance is selected. After the tuned configuration is obtained, the probability p_i of each operator i is set to $p_i = \frac{q_i}{\sum_{j=1}^{n} q_j}$.

Configuring AOS We further embarked the off-line algorithm configuration tools described above to fine-tune the hyper-parameter of the on-line AOS methods. The AOS parameters with their default parameter values and ranges for off-line configuration are listed in Table 1.

5 Experimental Results

5.1 The Static Operator Strategy

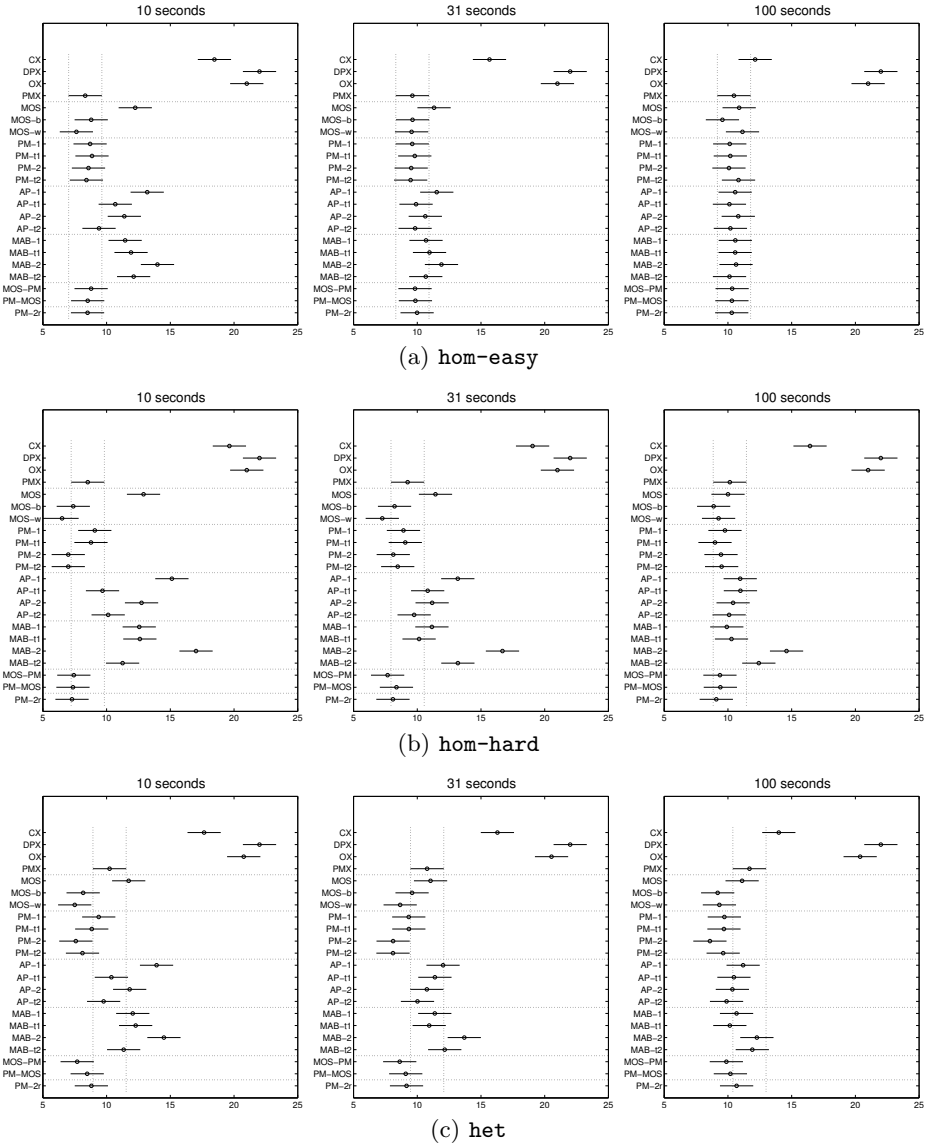
In each of the 9 training sets (three instance classes with three computation time), PMX is found to be the best performing operator, thus selected as the best off-line tuned static operator. Consider the 16 training instances of the `het` set, each with three stopping time, totaling 48 case studies. For each case study, we rank the four operators on each of the 10 runs and compare the their median rank. In the `het` set, PMX is best performing in 41 case studies, followed by CX in 4 case studies and OX in 3 case studies; in the `hom-easy` set, PMX performs best in 44 out of 48 case studies, and CX excels in the other four; PMX is most dominant in the `hom-hard` set, topping 47 case studies, while CX stands out in only one case studies. This shows PMX's dominance in the training set.

We further applied all the four static operators to the testing set, and their relative ranking performance in each of the 9 testing sets with a particular runtime is shown in the first block of each plot in Figure 1, and their performance across different runtime in each instance class is shown in Figure 2. For assessing the different candidate algorithms in the following, we test the statistical significance of each pairwise comparison by the Friedman test, and each plot in Figure 1 shows the median and the 95% simultaneous confidence intervals of candidate algorithm regarding these comparisons. If the intervals of two candidate algorithms overlap, then the difference between them is not statistically significant.[10] As clearly shown, PMX is dominantly best performing compared to the other three operators. It outperforms each of the other three operators statistically significantly (except few test cases in 100 seconds). CX, as the runner-up, significantly outperforms the other two operators except few cases in the `hom-hard` set. DPX turns out to the worst-performing candidate.

5.2 The Mixed Operator Strategy

The ranking performance of the three MOS based approaches is listed in the second block of each plot in Figure 1. Firstly, the two tuned versions MOS-b and MOS-w substantially improves over the default MOS-u with uniform probability in all case studies. The difference is statistically significant especially when the

[10] We further generated the box-plot of the median ranks across 10 trials of each instance, and the performance comparison in this median-rank box-plot and the presented confidence-interval plots are almost identical. The confidence-interval plot is shown here instead of median-rank box-plot since it displays additional information of statistical significance by the Friedman test.



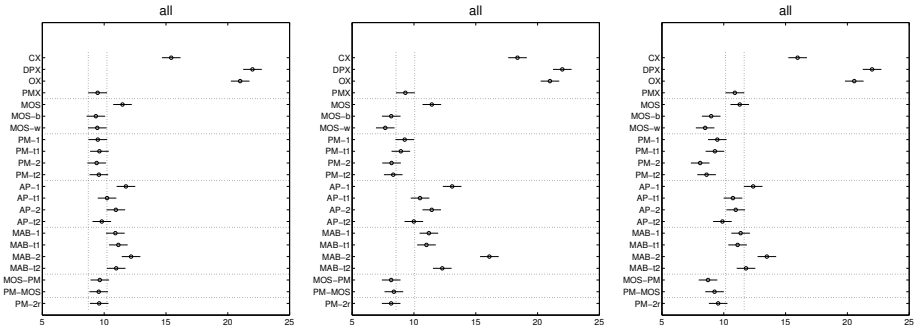
(a) `hom-easy`

(b) `hom-hard`

(c) `het`

Fig. 1. The ranking performance of different operator selection methods (acronyms see text) with different computation time 10, 31, and 100 seconds on (a) homogeneous easy (b) homogeneous hard or (c) heterogeneous instance set.

computation time is small, i.e. 10 or 31 seconds. MOS-w appears to be a slightly better way of tuning MOS compared to MOS-b, but the difference between them is never statistical significant. Both MOS-w and MOS-b perform better than the off-line tuned static operator PMX, especially when the instances are heteroge-



(a) `hom-easy` (b) `hom-hard` (c) `het`

Fig. 2. The ranking performance of different operator selection methods (acronyms see text) across different computation time on (a) homogeneous easy (b) homogeneous hard or (c) heterogeneous instance set.

neous as in the `het` set, and when the instances are hard as in the `hom-hard` set. In these two sets, the overall ranking difference between MOS-w and PMX across three computation times is significant. Even the untrained MOS-u can perform better than the trained static strategy PMX when solving `het` and `hom-hard` in 100 seconds. This interesting result indicates that, even if an operator that is dominantly better than the others exists, such as PMX in our case, varying the choice of operators at runtime can result in significantly better and more robust strategy than a static choice. This also sheds some light on how off-line configuration should be conducted: instead of finding one static configuration, varying the parameter values at runtime by a static distribution trained off-line may be a better idea. In fact, varying parameter values at runtime by a non-adaptive distribution is also applied to set the tabu list length in robust tabu search [31], a state-of-the-art algorithm for QAP.

5.3 The On-line Adaptive Operator Selection

The ranking performance of probability matching (PM), adaptive pursuit (AP), and multi-armed bandit (MAB) is illustrated in Figure 1 and 2, at the third, fourth, and fifth block, respectively. Within each block, the first two boxes refer to untrained and tuned version with the first reward function R_c^1 in Equation (2), while the latter two boxes with second reward function R_c^2 in Equation (3). It appears that the R_c^2 benefits PM and AP, while worsens MAB. In general, PM with R_c^2 (PM-2), is the best-performing adaptation method. The overall ranking differences between PM-2 and all the AP and MAB variants are significant for the heterogeneous instances (`het`) and hard instances (`hom-hard`).

The off-line configuration can improve the on-line adaptation methods when its quality is poor or when computation time is small. For example, the MAB-t2 significantly improves the performance of MAB-2 in terms of overall ranking as well as ranking in 10-second cases in the set of heterogeneous and hard instances.

Likewise, AP-t1 significantly improves AP-1. It appears that the performance of AP and MAB are more sensitive to their parameters and the reward function used; especially the scaling factor γ in MAB needs to be fine-tuned when the reward function is changed. So an off-line configuration should be helpful for these methods. However, the off-line configuration doesn't seem to be able to improve the performance of our best on-line method PM. Nevertheless, it is still recommended to use an off-line configuration, if one is uncertain about the algorithm performance, faces new problem domain or new setting of reward function, or the number of total operator generation is small.

Comparing with the static or mixed operator strategies, the ranking performance of on-line adaptation methods improve as computation time increases. Comparing with the off-line selected static operator PMX, PM-2 in general performs better, and the difference is significant when the instance set is heterogeneous as in `het`, and the computation time is long enough (100 seconds). It is interesting to see that even our best-performing on-line adaptation methods cannot outperform the fine-tuned mixed operator strategy. The difference between PM-2 and MOS-w and MOS-b is never significant. MOS variants tend to perform better in the homogeneous instances and at short or medium computation time, and PM-2 appears to be slightly better performing when the instances are heterogeneous and the computation time is long.

5.4 Combining MOS and AOS

We further investigate the possibility to incorporate both MOS and AOS together. The best MOS and AOS version found in this work, MOS-w and PM-2, respectively, are used for this study. Two ways of combination, namely, MOS-PM and PM-MOS are discussed below, and their results are presented in the second last block of each plot in Figure 1 and 2.

MOS-PM The first hybrid, named MOS-PM, is to tune the MOS-w parameters q_i (the quality vector to generate probability of choosing each operator) in the training set, and then use this to initialize the quality vector Q_i for each operator i in PM-2 by setting $Q_i = q_i / q_{\text{argmax}_i\, q_i}$. The scaling by setting the maximum initial Q_i to 1 is to make Q_i consistent with the magnitude in the reward function R_c^2. This approach amounts to a biased initial condition for the on-line adaptation method by an operator distribution trained off-line. However, MOS-PM does not bring any improvement to

- MOS-w, which shows that on-line adaptation cannot further improve a well-tuned non-adaptive mixed operator strategy. This agrees with the study in [16] that on-line adaptation methods cannot improve an off-line tuned static parameter configuration in ACO.
- PM-2 or PM-t2, which shows that either the fine-tuned initial quality Q_i does not interact well with the default setting of α and P_{min} in PM, or tuning initial condition for PM doesn't pay off in our context. We further ruled out the first factor by tuning the initial operator quality Q_i together

with α and P_{min}. However, no noticeable performance improvement can be observed comparing with tuning only α and P_{min} as in PM-t2. The same observation can be obtained on MAB and AP, where tuning initial condition doesn't improve adaptation performance, as long as its hyper-parameters for adaptation are already fine-tuned. This may be due to the large number of crossover operations in our experimental setting. There are already around 9 000 crossover generations in the 10-second case, where in each generation 20 crossover operations are performed, totaling 180 000 crossover operations. If the number of operations are low, such as when local search is applied, an off-line configuration of the initial conditions may pay off.

PM-MOS The second hybrid PM-MOS is to apply PM-2 on the training set to obtain the probability p_i of operator i for MOS-w. We first run PM-2 exhaustively on the training set (10 runs on each of the 16 training instances), keep track of the number of usage n_i^r for operator i in each run r, normalize it into the probability $p_i^r = n_i^r / \sum_i n_i^r$ of each operator i in run r, then derive p_i by averaging p_i^r over all training runs. This amounts to tuning parameters of MOS-w by an on-line adaptation method PM-2. In the study of [18], the on-line adaptation is found to perform worse than varying parameter setting randomly by the same empirical distribution in reactive tabu search [19]. If the same holds for PM-2, PM-MOS may perform better than PM-2. However, the results disagrees with our hypothesis. Comparing with PM-2, PM-MOS performs worse in the `het` set, and no observable difference on the two homogeneous instance set can be concluded. The reason for PM-MOS' inferior performance to PM-2 is further analyzed in the next section.

5.5 Further analysis on the effectiveness of on-line adaptation

The reason that PM-2 works better than PM-MOS on `het` set may be due to three factors: 1) the difference in the training and testing set induces experimental bias for the trained configuration; 2) the heterogeneity between instances, so that PM-2 can adapt to different settings for different instances; 3) PM-2, as an on-line adaptation method, has the ability to adapt the algorithm's behavior to local characteristics of the search space when running the algorithm for an instance [18]. We first took a look into the operator usage frequency in PM-2 on each instance on both the training and testing set. The operator usage frequency is actually very close from instance to instance, and no major difference between the training set and testing set can be observed. We set up experiments inspired by [18] to test the third factor as follows. For each run on each instance in the testing set, we keep track of the number of usage n_i of operator i in PM-2, and then randomly generate operators by MOS based on the empirical probability distribution of PM-2, $p_i = n_i / \sum_i n_i$. We allowed MOS to run exactly the same number of total operator generations in the PM-2. In such case, we observed that MOS may finish around 1% earlier than PM-2 due to the ease of computational overhead caused by the adaptation in PM-2. This result of the MOS run is

denoted as PM-2r as shown at the last block of each plot in Figure 1 and 2. Note that the empirical distribution in PM-2r is learned for each run on each instance, therefore, the first two factors above are ruled out. As shown in Figure 1 and 2, we observed that PM-2 and PM-2r have no performance difference in the two homogeneous instance sets `hom-easy` and `hom-hard`. However, in the `het` set of real-world benchmark QAP instances, PM-2 has a noticeable advantage over PM-2r, although the difference is not yet statistically significant. This indicates that the best adaptive operator selector in our context, PM-2, does adapt well to the local characteristics of the search space when running on the real-world benchmark QAP instances, however, it fails to do so for the generated instances with more random structures.

6 Conclusions and Future Works

In this work, we provide an empirical study of off-line parameter configuration and on-line parameter adaptation on the operator selection problem in evolutionary algorithm. We extended [17] by incorporating off-line configuration with the non-static operator selection methods, including: i) a non-adaptive mixed operator strategy (MOS), which assigns a probability distribution for selecting each operator; ii) three adaptive operator selection (AOS) methods Probability Matching, Adaptive Pursuit, and Multi-Armed Bandit. State-of-the-art off-line algorithm configuration tools are applied to this end, including iterated racing [7] and post-selection techniques [30]. One major contribution in this study is to identify an automatically tuned MOS as one of the best performing approaches for operator selection. The results show that even when a dominantly best choice of static operator exists, using an automatically tuned operator probability distribution still significantly outperforms the best static operator approach. This also sheds some light to the future design of off-line algorithm configuration: instead of tuning for a static parameter configuration, it may be a better idea to tune a distribution from which the parameter configurations are randomly generated and changed during algorithm run. Besides, we also improved the performance of on-line AOS methods by considering different reward function and an off-line configuration of its hyper-parameters.

Our future works aim to extend this study to operator selection problem other than only crossover operators: local search operators, mutation operators, selection criteria operators, etc., or even a combination of different kinds of operators to test the scalability of the approaches studied in this work. We also plan to include other state-of-the-art operator selection techniques such as Dynamic Multi-Armed Bandit (DMAB) [32]. Another interesting direction is to do a portfolio-based operator selection by taking into account also the instance features. Since adapting operator choice according to local characteristics of search space is found to be crucial for the good performance of on-line method PM-2, applying Markov Decision Process [33] by translating the local landscape characteristics into different states at each time step and performing state-based on-line learning becomes a good direction to follow.

Acknowledgments

We sincerely thank Dr. Thomas Stützle for sharing the QAP instance generator, and for the insightful discussions on the instances and the result presentation. This work was partially supported by the BMBF Verbundprojekt E-Motion.

References

1. Hamadi, Y., Monfroy, E., Saubion, F., eds.: Autonomous Search. Springer, Berlin, Germany (2007)
2. Hoos, H.H.: Programming by optimization. Communications of the ACM **55**(2) (2012) 70–80
3. Battiti, R., Brunato, M., Mascia, F.: Reactive search and intelligent optimization. Springer, New York (2008)
4. Birattari, M.: Tuning Metaheuristics: A machine learning perspective. Springer, Berlin, Germany (2009)
5. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: Portfolio-based algorithm selection for sat. J. Artif. Intell. Res.(JAIR) **32** (2008) 565–606
6. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In Langdon, W.B., et al., eds.: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers, San Francisco, CA (2002) 11–18
7. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-Race and iterated F-Race: An overview. In Bartz-Beielstein, T., et al., eds.: Experimental Methods for the Analysis of Optimization Algorithms. Springer, Berlin, Germany (2010) 311–336
8. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research **36** (2009) 267–306
9. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of solvers. In Gent, I.P., ed.: Principles and Practice of Constraint Programming – CP 2009. Volume 5732 of Lecture Notes in Computer Science. Springer (2009) 142–157
10. Hutter, F., Bartz-Beielstein, T., Hoos, H.H., Leyton-Brown, K., Murphy, K.: Sequential model-based parameter optimisation: an experimental investigation of automated and interactive approaches. In Bartz-Beielstein, T., et al., eds.: Empirical Methods for the Analysis of Optimization Algorithms. Springer, Berlin, Germany (2010) 363–414
11. Yuan, Z., de Oca, M.M., Birattari, M., Stützle, T.: Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. Swarm Intelligence **6**(1) (2012) 49–75
12. Angeline, P.J.: Adaptive and self-adaptive evolutionary computations. In Palaniswami, M., et al., eds.: Computational intelligence: a dynamic systems perspective, IEEE Press (1995)
13. Eiben, A.E., Michalewicz, Z., Schoenauer, M., Smith, J.E.: Parameter Control in Evolutionary Algorithms. In Lobo, F.G., Lima, C.F., Michalewicz, Z., eds.: Parameter Setting in Evolutionary Algorithms. Studies in Computational Intelligence Series. Springer, Berlin, Germany (2007) 19–46
14. Lobo, F., Lima, C.F., Michalewicz, Z., eds.: Parameter Setting in Evolutionary Algorithms. Springer, Berlin, Germany (2007)

15. Fialho, Á., Schoenauer, M., Sebag, M.: Toward comparison-based adaptive operator selection. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation, ACM (2010) 767–774
16. Pellegrini, P., Stützle, T., Birattari, M.: A critical analysis of parameter adaptation in ant colony optimization. Swarm Intelligence **6**(1) (2012) 23–48
17. Francesca, G., Pellegrini, P., Sttzle, T., Birattari, M.: Off-line and on-line tuning: A study on operator selection for a memetic algorithm applied to the qap. In Merz, P., Hao, J.K., eds.: Proc. of EvoCOP. Volume 6622 of Lecture Notes in Computer Science., Springer (2011) 203–214
18. Mascia, F., Pellegrini, P., Birattari, M., Stützle, T.: An analysis of parameter adaptation in reactive tabu search. International Transactions in Operational Research (2013) To appear.
19. Battiti, R.: The reactive tabu search. ORSA Journal on Computing (1994)
20. Pardalos, P.M., Wolkowicz, H., eds.: Quadratic Assignment and Related Problems. DIMACS Series. American Mathematical Society (1994)
21. Merz, P., Freisleben, B.: Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. IEEE Transaction on Evolutionary Computation **4**(4) (2000) 337–352
22. Krempser, E., Fialho, Á., Barbosa, H.J.C.: Adaptive operator selection at the hyper-level. In Coello, C., et al., eds.: Prof. of PPSN. Volume 7492 of Lecture Notes in Computer Science., Springer (2012) 378–387
23. Fialho, Á., Da Costa, L., Schoenauer, M., Sebag, M.: Extreme value based adaptive operator selection. In: Parallel Problem Solving from Nature–PPSN X, Springer (2008) 175–184
24. Corne, D.W., Oates, M.J., Kell, D.B.: On fitness distributions and expected fitness gain of mutation rates in parallel evolutionary algorithms. In: Proc. of PPSN VII, Springer (2002) 132–141
25. Thierens, D.: An adaptive pursuit strategy for allocating operator probabilities. In: Proc. of IEEE CEC, IEEE (2005) 1539–1546
26. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine learning **47**(2) (2002) 235–256
27. Burkard, R.E., Karisch, S.E., Rendl, F.: Qaplib–a quadratic assignment problem library. Journal of Global Optimization **10**(4) (1997) 391–403
28. Stützle, T., Fernandes, S.: New benchmark instances for the qap and the experimental analysis of algorithms. In: Proc. of EvoCOP. Volume 3004 of LNCS. Springer (2004) 199–209
29. McGeoch, C.: Analyzing algorithms by simulation: variance reduction techniques and simulation speedups. ACM Computing Surveys (CSUR) **24**(2) (1992) 195–212
30. Yuan, Z., Stützle, T., Montes de Oca, M.A., Lau, H.C., Birattari, M.: An analysis of post-selection in automatic configuration. In: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference, ACM (2013) 1557–1564
31. Taillard, E.: Robust taboo search for the quadratic assignment problem. Parallel computing **17**(4) (1991) 443–455
32. Fialho, A., Costa, L., Schoenauer, M., Sebag, M.: Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. In Stützle, T., et al., eds.: Proc. of LION, Berlin, Germany, Springer-Verlag (2009) 176–190
33. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. 1st edn. John Wiley & Sons, Inc., New York, NY, USA (1994)