Selection of Auxiliary Objectives Using Landscape Features and Offline Learned Classifier

Anton Bassin(✉) and Arina Buzdalova

ITMO University, 49 Kronverksky Pr., Saint-Petersburg, Russia 197101
anton.bassin@gmail.com, abuzdalova@gmail.com

Abstract. In order to increase the performance of an evolutionary algorithm, additional auxiliary optimization objectives may be added. It is hard to predict which auxiliary objectives will be the most efficient at different stages of optimization. Thus, the problem of dynamic selection between auxiliary objectives appears. This paper proposes a new method for efficient selection of auxiliary objectives, which uses fitness landscape information and problem meta-features. An offline learned meta-classifier is used to dynamically predict the most efficient auxiliary objective during the main optimization run performed by an evolutionary algorithm. An empirical evaluation on two benchmark combinatorial optimization problems (Traveling Salesman and Job Shop Scheduling problems) shows that the proposed approach outperforms similar known methods of auxiliary objective selection.

Keywords: Evolutionary algorithms · Multi-objective optimization · Auxiliary objectives · Fitness landscape features

1 Introduction

Evolutionary algorithms (EAs) are generic meta-heuristic optimization algorithms. An EA searches solution candidates based on the current state whilst previously reached historical states are not taken into account during the runtime. The information derived from the fitness landscape and from the optimization problem instance may be used to determine the state of an evolutionary algorithm. In these terms, the optimization problem transforms into searching the EA states which correspond to global optima on the fitness landscape. Auxiliary objectives may be used instead of the target objective or along with the target objective. Auxiliary objectives serve to multi-objectivise a single objective problem. In some cases this transformation may increase efficiency of an EA.

An auxiliary objective is efficient if its usage leads to decrease of the time needed to find the optimum of the target objective. Different auxiliary objectives have different efficiency on various stages of optimization. For example, at the stagnation point of the EA the most aggressive auxiliary objective may move the optimization process away from getting stuck in a local optimum. Inversely,

B. Hu and M. López-Ibáñez (Eds.): EvoCOP 2017, LNCS 10197, pp. 173–188, 2017.
DOI: 10.1007/978-3-319-55453-2_12

if the current algorithm state corresponds to the situation where solution candidates are located near the global optimum, we would want to use less aggressive auxiliary objectives.

At present time, researchers are looking for new ways of parameterizing fitness landscape features and applying received techniques in the evolutionary computation field [1,2]. Authors of paper [3] suggested a method to multi-objectivise single objective problems by using an elementary landscape decomposition of their objective function. However, to the best of our knowledge, landscape features have never been used to guide dynamical selection of auxiliary objectives. The existing objective selection algorithms use random selection based on the number of iterations [4] or reinforcement learning based on differences in target objective values [5].

One of the first approaches to transform a single-objective problem into a multi-objective one was proposed by Knowles et al. in [6]. The authors suggest decomposing the target objective into several components, which should be independent. The decomposed objectives are optimized simultaneously. Another method belongs to Jensen [4]. The idea is to use auxiliary objectives in combination with the target objective. Furthermore, the auxiliary objectives used in paper [4] are changed dynamically. The author concluded from the obtained experimental results that using one auxiliary objective at a time is the best approach, but he also underlined questions on when to change the auxiliary objective and to which objective it should be changed. There also exists an adaptive auxiliary objective selection method based on reinforcement learning called EA+RL [5]. The main idea is to use the reinforcement learning to train online (during the EA runtime) an agent, which tries to predict the most efficient auxiliary objective on each evolutionary iteration. The aforementioned method was improved by Petrova et al. in [7].

We tested the efficiency of the method that we propose in the present paper on multiple instances of two benchmark problems: The Traveling Salesman Problem (TSP) and The Job Shop Scheduling Problem (JSSP). However, the proposed method is not designed for solving any specific optimization problem, it is a general approach for selection of the most efficient auxiliary objective during EA runtime. Therefore, we compared the proposed method with other approaches of objective selection. Additionally, to confirm the reliability of the obtained results we checked the corresponding values for statistical distinguishability.

The rest of the paper is organized as follows. In Sect. 2 discusses the main aspects of the proposed method of auxiliary objective selection. Section 3 presents experiment results of solving TSP. Section 4 presents the results for the JSSP, and we conclude in Sect. 5.

2 The OLHP Method

We propose a new auxiliary objective selection strategy named *The Offline Learned Helper Picker (OLHP)*. The term *Offline* is used because the meta-classifier for auxiliary objective selection is trained offline with machine learning methods. The learning dataset is gathered from the training EA runs on training instances of an optimization problem. The learning dataset vector contains

properties of the problem instance and feature values of the fitness landscape of the current EA population. The term *Helper* is used as a synonym to "auxiliary objective". The OLHP consists of two main phases: the meta-classifier learning and objective selection during the EA runtime. The implementation of OLHP, along with experimental evaluation results, is available at Bitbucket repository[1].

2.1 Learning the Meta-Classifier Phase

The meta-classifier is learned only once for each optimization problem T. Input parameters of this phase are: **1.** A set of training problem instances $L \subset T$, L consists of any $\ell \in T$, where T is the set of all possible instances of a particular optimization problem. **2.** A set of rules for auxiliary objectives generation $G = \{g(i)\}, g(i) = h_i : \mathbb{N} \to H$, where H is the set of all possible auxiliary objectives for the considered optimization problem.

At the considered phase we construct the dataset of learning samples and then build the meta-classifier. To do this we need to perform n runs of the EA for each training problem instance. The value of the constant n may be manually tuned to find better meta-classifier metrics.

The current EA state is described by two components: static meta-features of the problem instance and features of the target objective landscape at the current population, which are extracted dynamically during the runtime. From each particular EA state st_j we make k runs ($k = |H_k|$ is the total number of used auxiliary objectives, $G(H) = H_k = \{h_i\}$ is a set of generated auxiliary objectives) in parallel for $n_{\mathrm{iter}} = \frac{I_{\max}}{k}$ iterations, where $I_{\max}$ corresponds to the maximum number of iterations in a training run of the EA. Thus, the maximum number of considered EA states is $j = k$. Accordingly, there is a chance for each auxiliary objective to be picked at each EA state. All parallel EA threads optimize different auxiliary objective function h_i simultaneously with the target objective. After performing latter evaluations we can make an assumption on which auxiliary objective h_i would be the most efficient if using it from the EA state st_j for n_{iter} EA iterations.

We identify an efficiency of an auxiliary objective by comparing the values of target objective for the best solution candidate in the beginning of each EA thread and in the end of its work. If several threads showed an equal increase in the target fitness value of the best solution, then the best auxiliary objective for the EA state st_j is selected from the first thread. Thereby, we have one learning dataset vector which is comprised of the meta-features of the problem instance and the fitness landscape features corresponding to the EA state st_j. The target value (or class value) of this vector is number i, which corresponds to the most efficient auxiliary objective h_i.

After performing the training evaluations from the EA state st_j, we move forward to a new state st_{j+1}. As the state st_{j+1} we pick an EA state after using the most efficient auxiliary objective h_j. The steps described above are processed until the maximum number of EA iterations $I_{\max}$ is reached. The final thing to

[1] https://bitbucket.org/BASSIN/2017-olhp-tsp-jssp/src.

do is to train and save the meta-classifier for selection of auxiliary objectives. Algorithm 1 presents the pseudocode of the meta-classifier learning phase.

Algorithm 1. Learning of the meta-classifier

```
1: procedure LearnSelector(Insts_tr, G)        ▷ Insts_tr is a set of training instances, G - set of auxiliary objective generating rules
2:     r ← the number of runs for each training instance
3:     for all tr in Insts_tr do
4:         metaFeatures ← extractMetaFeatures(tr)
5:         H_k ← G(tr)                          ▷ Generate auxiliary objectives for train problem instance tr
6:         i ← 0
7:         while i < r do
8:             I_max ← calculateMaxIterationNumber(tr)
9:             n_iter ← I_max / |H_k|           ▷ Setting iterations number between algorithm states
10:            EA.initialize(tr)                ▷ Initialize evolutionary algorithm with train instance tr
11:            j ← 0
12:            while j < I_max do
13:                st_j ← saveState(EA, metaFeatures)
14:                for all h in H_k do          ▷ For each auxiliary objective
15:                    EA.run(n_iter, h)        ▷ Run EA with auxiliary objective h for n_iter iterations
16:                    fitnessRaise_h ← calculateFitnessDiff(EA)
17:                    st_h ← saveState(EA)
18:                end for
19:                h_best ← findBestHelper(∀fitnessRaise_h)   ▷ Identify the most efficient auxiliary objective
20:                dataset.put(h_best, st_j.getMetaFeatures(), st_j.getLandscapeFeatures())
21:                EA.setState(st_h_best)       ▷ Set EA state to the state after using the best auxiliary objective
22:                j ← j + n_iter
23:            end while
24:            i ← i + 1
25:        end while
26:    end for
27:    classifier.train(dataset)        ▷ Learn the meta-classifier for objective selection
28:    classifier.serialize()           ▷ Serialize trained model for future usages
29: end procedure
```

2.2 Objective Selection During the EA Runtime

This subsection describes the algorithm that dynamically selects and applies auxiliary objectives during EA runtime. The input parameters for this $OLHP$ phase are: **1.** An instance of the optimization problem we want to solve $\ell \in T$, where T is a space of all possible instances of a particular optimization problem; **2.** A set of rules for generation of auxiliary objectives $G = \{g(i)\}, g(i) = h_i$:

$\mathbb{N} \rightarrow H$, where H is the set of all possible auxiliary objective functions we are working with. **3.** A meta-classifier learned beforehand on problem instances from T: predict$(v) = k, g(k) = h_{\text{best}}$, where v is the vector of fitness landscape features of the current population and static meta-features of the problem instance.

The first step of this phase is initialization of all required structures and deserialization of the meta-classifier for considered optimization problem. At the next step, we start an evolutionary algorithm. Each $n_{\text{iter}} = \frac{I_{\max}}{k}$ $(k = |H_k| = |G(H)|)$ iteration period we predict the most efficient auxiliary objective basing on the problem instance static features and fitness landscape features of the current EA population. The predicted objective h_{best} is used by the EA simultaneously with the target objective for the next n_{iter} iterations. Restriction on optimization by only one additional objective is made for making it possible to learn and solve many problem instances of various sizes for a reasonable computational time. Algorithm 2 presents the detailed pseudocode for this OLHP phase.

Algorithm 2 Solving the problem instance

```
1: procedure SolveProblem(inst, G, cl)        ▷ inst is a problem instance to optimize, cl – learned classifier
2:     cl.deserialize()
3:     metaFeatures ← extractMetaFeatures(inst)
4:     H_k ← G(inst)                          ▷ Generate auxiliary objectives for the problem instance
5:     I_max ← calculateMaxIterationNumber(inst)
6:     n_iter ← I_max / |H_k|                 ▷ Setting iterations number between algorithm states
7:     EA.initialize(inst)                    ▷ Initialize EA with optimization problem instance
8:     j ← 0
9:     while j < I_max do
10:        if j mod n_iter = 0 then           ▷ Time for switching an auxiliary objective
11:            st_j ← getState(EA, metaFeatures, getLandscapeFeatures(EA))
12:            h_predicted ← cl.predict(st_j)
13:            EA.setHelper(h_predicted)
14:        end if
15:        EA.runIteration()                  ▷ Make one iteration of EA
16:        j ← j + 1
17:    end while
18:    EA.saveBestSolution()                  ▷ Save the optimization result
19: end procedure
```

2.3 Fitness Landscape Features

We use generic fitness landscape features of an EA population which are eligible for almost any optimization problem. A population of size p may be represented as a set of random variables $P = \{s_i\}, i = 1..p$. On each EA iteration we can obtain values of random variables s_i. Hence, we can calculate statistical metrics of set P.

Since we are solving meta-classification task for any instance of optimization problem T, we need to normalize the values of received landscape features. For such a normalization, we propose to use the ratio of the target objective value on current solution candidate to the best known solution candidate value: $\frac{G(s_i)}{G(s_{\text{best}})} = x_i$, where $G(s)$ is the target objective function. The set of random variables $\{x_i\} = P_{\text{normalized}}$ is normalized for all instances of problem T. Thus, we suggest to use the following statistical metrics calculated on the set $P_{\text{normalized}}$ as fitness landscape features:

1. Med – the median value.
2. $\bar{x} = \frac{1}{p}\sum_{i=1}^{p} x_i$ – the arithmetical mean.
3. $H_{\text{mean}} = \frac{p}{\sum_{i=1}^{p}\frac{1}{x_i}}, x_i > 0$ – the harmonic mean.
4. Dev – the standard deviation.
5. $Q_{\text{mean}} = \sqrt{\frac{1}{p}\sum_{i=1}^{p}(x_i - \bar{x})^2}$ – the sample variance.

3 Applying OLHP to Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a classic NP-hard problem in combinatorial optimization. Each TSP instance may be described by a set of cities $\{c_i\}, \forall i = 1 \ldots N$ and a distance matrix M of size $N \times N$. Elements of M represent the distance between a pair of cities. For example, $M(c_i, c_j)$ is the distance between the cities c_i and c_j. The TSP asks the following question: "What is the shortest possible route that visits each city once and returns to the origin city?". In other words, we need to find a Hamiltonian path with the lowest total distance. For the path vector $\rho = (\rho_1, \rho_2, \ldots, \rho_N)$ we can calculate the total distance cost using (1):

$$D(\rho) = \sum_{i=1}^{N} M(c_{\rho_{[i]}}, c_{\rho_{[i\oplus 1]}}),$$
$$\text{where } i \oplus 1 = \begin{cases} i+1, & \text{if } i < N \\ 1, & \text{if } i = N \end{cases} \tag{1}$$

In experimental evaluations, we use symmetric TSP problem instances. In the symmetric TSP problem, the value of a path from one city to another is equal to the value of the reverse path: $M(c_i, c_j) = M(c_j, c_i)$. More detailed explanation of the TSP may be found in [8].

3.1 TSP Meta-Features

We need to specify TSP meta-features which we would use as a part of machine learning vector. Meta-features should contain information, which represents the properties of a particular TSP instance. The following features meet this requirement:

1. V_{num} – the number of cities.
2. E_{min} – the minimum distance between a pair of cities.
3. E_{max} – the maximum distance.
4. E_{avg} – the average distance.
5. E_{med} – the median distance.
6. Dev_E – the standard deviation of distances.
7. $\text{QE}_{-\text{avg}}$ – the number of distances shorter than E_{avg}.
8. $\text{Sum}_{\text{min}_E}$ – the sum of V_{num} minimal distances.

The latter TSP meta-features were successfully used by Kanda et al. in [9] to classify Traveling Salesman Problems and in [10] to recommend meta-heuristics for solving TSP.

3.2 TSP Auxiliary Objectives Generation

In the OLHP method, an auxiliary objective is required to have some property, which depends only on the problem instance, but not on the individual or the iteration number. Unfortunately, the existing approaches of auxiliary objective generation [4,6] do not provide us objectives with such kind of a property, because in these approaches objectives are generated using randomly picked cities.

To generate auxiliary objectives which depend only on the problem instance, we propose a new method of auxiliary objective generation inspired by the k-nearest neighbor classification algorithm [11].

In [4], the following auxiliary objective function was proposed:

$$h(\rho, s) = \sum_{i=1}^{|s|} (M(c_{\rho[\rho^{-1}[s[i]\ominus 1]}, c_{s[i]}) + M(c_{s[i]}, c_{[\rho^{-1}[s[i]]\oplus 1]}), \tag{2}$$

where s is the subset of the set of cities $C = \{1, 2, \ldots, N\}$, $\rho^{-1}(x)$ is the position of x in ρ, $\ominus 1$ is the reverse operator to $\oplus$ 1.

In our approach, we use Eq. (2) to generate auxiliary objectives. We generate subsets of cities (the s parameter in (2)) by partitioning the set of cities $C = \{c_i\}, i = 1 \ldots N$ using the following algorithm:

1. Sort the set of all cities C by the following criteria:

$$\text{Knn}(C) \rightarrow S_C = \{c_1, c_2, \ldots, c_N\}, \tag{3}$$

where $\text{Knn}(C)$ is the sorting operator, S_C – is the ordered set.
For each pair of elements from S_C:

$$c_i \preceq_{\text{knn}} c_j, \tag{4}$$

where $i, j = 1 \ldots N$ and the relation $\preceq_{\text{knn}}$ is true when the total distance from city c_i to k nearest neighbor cities is less or equal to the corresponding total distance for the city c_j.

2. Divide the ordered set S_C into subsets $s_j \subset S_C, j = 1 \dots r$ of equal cardinality. The number of elements in the last subset s_r may be less than the number of elements in the other subsets.

Note that $s_i \cap s_j = \emptyset$, $\forall i, j = 1 \dots r, i \neq j$. This fact means that the generated auxiliary objectives $h(\rho, s_j)$ have disjoint properties on a problem instance. This should make possible for each auxiliary objective to be the most efficient objective at different stages of optimization process.

3.3 Experimental Evaluation on TSP

We compare the OLHP method with the following approaches of optimizing the target objective with auxiliary objectives. First, we consider the method proposed by Jensen [4], where the auxiliary objective is dynamically reselected after a fixed number of EA iterations. In the second considered approach [7], named Multi-Objective Evolutionary Algorithm + Reinforcement Learning (MOEA+RL), RL agent learns to select auxiliary objectives during the EA runtime and non-stationarity of the environment is taken into account. The last considered approach is a modified combination of two known algorithms. Two auxiliary objectives are composed in the manner described by Jähne et al. in [12] and the first selected auxiliary objective is switched to another one at the half of the EA runtime as suggested in [4].

The TSP instances for the experimental evaluation were taken from TSPLIB[2] library. The crossover and mutation operators were identical to the corresponding operators from the papers mentioned above. Also, the *2-opt* local search heuristic [13] was used.

The crossover probability was equal to 40%. The population size was $P_{\text{size}} = 100$. The limit of target objective evaluations for each EA run was calculated in the manner proposed in [12]: $\text{ev}_{\max}(N, m) = \sqrt{N^3} * m$, where N is the total number of cities, m is a manually chosen parameter (in our experiments we used $m = 10$). The number of EA runs for each training problem instance was $n = 4$.

The auxiliary objectives for the methods proposed by Jensen and Jähne were generated using the rules, which are provided in the related papers. The results for both the OLHP and MOEA+RL algorithms were obtained using the same auxiliary objectives, namely the *K-nn* auxiliary objectives proposed in Sect. 3.2. We used the following parameters to generate auxiliary objectives: $k = 5$, $r = 5$.

The comparison with MOEA+RL was intended to evaluate the efficiency of the proposed objective selection scheme without the influence of the objective generation approach, as the both OLHP and MOEA+RL algorithms are in equal conditions in terms of the used auxiliary objectives. At the same time, in the Jensen and Jähne/Jensen methods, the original auxiliary objectives from the corresponding works were applied, so the comparison with these methods was performed to evaluate the efficiency of the entire proposed approach for the TSP optimization.

[2] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95.

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) [14] was used as the base evolutionary algorithm in all experiments. The program code was written in Java and Groovy languages. The following frameworks were used: Watchmaker[3] – for evolutionary computations, Weka[4] – for the OLHP machine learning operations.

Learning of Meta-Classifier. The Random Forest classifier was used for building the OLHP auxiliary objective selector. The Random Forest parameters had default[5] values from the Weka framework.

To obtain train and test sets, TSP instances from the TSPLIB were divided into subsets. We were guided by the idea that train and test instances should have items with similar meta-properties. The train TSP instances are listed in Appendix A.

To estimate how accurately our predictive model would perform in practice, we cross-validated our classifier. For the same limitation on similarity of train and test problem instances we were forced to use a special set of problems for cross-validation. We made 10-fold cross-validation on instances listed in Appendix A. The performance metric values of our classification model on TSP after the cross-validation were: Estimated Error Rate = 0.16, Precision = 0.90, Recall = 0.96, F-measure = 0.93. The latter values confirm that there exists a correlation between the EA state features and selection of the most efficient auxiliary objective.

Results of Solving TSP. We used 44 TSP instances to perform experiments. Further, the final solution results for each method of auxiliary objective selection were obtained $\eta = 40$ times and averaged.

Table 1 shows mean and standard deviation of the best obtained value of the target objective for the OLHP, MOEA+RL, Jensen and Jensen/Jähne methods. Cells with the best values are marked with bold text. The last row of Table 1 shows the total number of instances, on which the particular method has outperformed other approaches. Note that the sum of the values in the last row is not equal to the number of total considered test instances. It is explained by the situation when several methods showed the best mean target value. In this case, we increment the corresponding counters for each such method. To summarize, it can be concluded from Table 1 that the newly proposed OLHP method outperformed the considered approaches of auxiliary objective selection on the set of test TSP instances.

Statistical Testing. According to [15], we used the Wilcoxon signed-rank test to detect significant differences in behavior of two algorithms. The pairwise statistical test was applied on the average results obtained on test instances for each pair of the considered approaches. In order to perform multiple comparisons and

[3] http://watchmaker.uncommons.org.
[4] http://www.cs.waikato.ac.nz/ml/weka.
[5] http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/RandomForest.html.

Table 1. Mean and standard deviation of resulting fitness (TSP)

Problem	Best	OLHP	MOEA+RL	Jensen	Jen/Jäh
a280	2579	2597.9 ± 13.1	2599.4 ± 13.4	2597.2 ± 16.3	**2597.2 ± 11.8**
ali535	202339	204624.4 ± 1587.0	**204382.0 ± 1482.5**	205909.5 ± 1505.1	204853.0 ± 1463.7
att48	10628	**10628.5 ± 2.2**	10629.1 ± 4.9	10631.4 ± 6.5	10645.1 ± 17.7
bier127	118282	118337.3 ± 116.0	118436.2 ± 259.4	**118320.6 ± 83.0**	118358.1 ± 187.9
brg180	1950	1952.8 ± 4.5	1952.5 ± 4.3	**1950.0 ± 0.0**	**1950.0 ± 0.0**
ch150	6528	6548.0 ± 12.4	6550.8 ± 12.7	**6544.4 ± 12.6**	6547.6 ± 12.9
d1291	50801	51585.9 ± 210.2	51594.4 ± 240.7	51592.7 ± 191.1	**51560.5 ± 270.5**
d657	48912	49292.3 ± 151.9	49353.6 ± 141.5	**49285.6 ± 127.6**	49358.0 ± 131.0
dsj1000	18659688	**18860887.9 ±45886.0**	18870762.9 ±50593.3	18861694.6 ±60582.9	18887428.3 ±61706.0
eil76	538	**544.4 ± 0.0**	544.5 ± 0.5	544.7 ± 0.9	544.4 ± 0.1
fl417	11861	11927.0 ± 5.2	**11926.8 ± 6.0**	11942.1 ± 13.9	11939.7 ± 22.4
gr137	69853	**69862.6 ± 28.5**	69878.8 ± 57.9	69878.8 ± 44.4	69864.0 ± 29.3
gr229	134602	135130.0 ± 400.5	135087.3 ± 346.0	**135048.9 ± 346.7**	135051.8 ± 267.1
gr666	294358	297940.3 ± 1536.3	298189.9 ± 1454.3	**297663.3 ± 1336.9**	297826.6 ± 1426.5
gr96	55209	**55305.2 ± 58.1**	55312.6 ± 77.0	55352.1 ± 60.6	55326.9 ± 70.2
kroA100	21282	21287.0 ± 9.5	21287.0 ± 10.0	21287.0 ± 10.0	**21285.4 ± 0.0**
kroA200	29368	**29393.5 ± 59.2**	29398.2 ± 67.6	29426.1 ± 111.9	29395.7 ± 62.3
kroB100	22141	**22140.9 ± 11.1**	22148.2 ± 28.7	22141.3 ± 13.3	22145.9 ± 21.6
kroB150	26130	**26148.2 ± 49.5**	26190.0 ± 64.9	26162.5 ± 60.7	26149.8 ± 51.1
kroC100	20749	**20750.8 ± 0.0**	**20750.8 ± 0.0**	**20750.8 ± 0.0**	20750.8 ± 0.5
kroD100	21294	**21311.2 ± 30.1**	21333.5 ± 45.0	21374.0 ± 31.8	21314.5 ± 33.2
lin105	14379	**14383.0 ± 0.0**	**14383.0 ± 0.0**	**14383.0 ± 0.0**	**14383.0 ± 0.0**
lin318	42029	42321.6 ± 175.3	42323.7 ± 180.6	42314.6 ± 154.1	**42297.7 ± 167.8**
pcb1173	56892	57909.2 ± 166.2	57982.5 ± 250.3	**57885.9 ± 199.6**	57909.2 ± 201.1
pcb442	50778	**51263.1 ± 132.1**	51305.7 ± 199.0	51312.8 ± 202.1	51307.3 ± 167.3
pr1002	259045	**262779.5 ± 868.7**	263100.1 ± 1115.0	263427.5 ± 786.9	263178.0 ± 942.0
pr107	44303	**44328.9 ± 39.9**	44337.7 ± 50.8	44372.1 ± 74.5	44336.6 ± 50.0
pr124	59030	**59030.7 ± 0.0**	**59030.7 ± 0.0**	59032.9 ± 9.6	**59030.7 ± 0.0**
pr144	58537	**58535.2 ± 0.0**	58536.1 ± 5.2	58562.5 ± 14.8	58565.3 ± 19.9
pr152	73682	73697.4 ± 41.3	73711.2 ± 55.0	73783.6 ± 61.2	**73691.6 ± 49.9**
pr226	80369	80374.6 ± 12.4	**80374.4 ± 12.4**	80382.9 ± 38.8	80385.8 ± 29.7
pr299	48191	**48320.1 ± 111.8**	48370.7 ± 165.5	48434.7 ± 210.8	48398.6 ± 134.6
pr439	107217	107600.7 ± 424.3	**107597.9 ± 354.4**	107666.0 ± 498.6	107875.6 ± 510.4
rat195	2323	**2340.6 ± 5.5**	2344.9 ± 6.3	2343.1 ± 5.2	2342.9 ± 6.1
rat783	8806	**8932.3 ± 23.8**	8949.3 ± 25.4	8946.2 ± 25.5	8941.6 ± 22.8
rd100	7910	7912.3 ± 8.3	**7911.9 ± 5.4**	7914.7 ± 12.5	7914.7 ± 11.4
rd400	15281	15394.0 ± 54.6	15428.7 ± 58.8	15386.8 ± 54.3	**15377.9 ± 55.8**
si1032	92650	**92650.0 ± 0.2**	92656.6 ± 19.8	92720.3 ± 43.9	92673.4 ± 22.2
si535	48450	48496.4 ± 17.4	48496.2 ± 22.6	48543.2 ± 33.1	**48487.8 ± 16.2**
tsp225	3916	3876.3 ± 21.1	3877.8 ± 21.1	3873.1 ± 21.5	**3869.2 ± 16.8**
u1060	224094	226731.1 ± 671.0	**226627.1 ± 721.0**	226825.6 ± 690.4	226962.5 ± 687.4
u159	42080	**42075.7 ± 0.0**	**42075.7 ± 0.0**	**42075.7 ± 0.0**	**42075.7 ± 0.0**
u724	41910	42298.9 ± 137.3	42337.1 ± 119.3	**42247.3 ± 96.4**	42280.5 ± 121.7
vm1084	239297	241691.7 ± 962.0	241953.6 ± 898.0	**241432.7 ± 704.0**	241822.2 ± 770.0
Total best		**21**	10	12	12

control the family-wise error rate we adjusted the obtained p-values by using the Holm–Bonferroni correction method.

The *adjusted p-values* for pairs of methods of auxiliary objective selection were the following: OLHP – MOEA+RL = 1.0e-03, OLHP – Jensen = 4.2e-02, OLHP – Jensen/Jähne = 4.2e-02. Therefore, the OLHP method shows a significant improvement over MOEA+RL, Jensen and Jensen/Jähne approaches, with the level of significance $\alpha = 0.05$.

4 Applying OLHP to Job Shop Scheduling Problem

The OLHP method was applied to the Job Shop Scheduling Problem (JSSP) for further verification of its efficiency. Similarly to TSP, JSSP is a well-known NP-hard combinatorial optimization problem.

A JSSP instance of size $n \times m$ consists of n jobs $\{J_1, J_2, \ldots, J_n\} = J$ and m machines $\{M_1, M_2, \ldots, M_m\} = M$. Each job J_i contains a sequence of m operations $(o_{i1}, o_{i2}, \ldots, o_{im})$. Jobs and machines have mutual constraints, because an operation o_{ij} may be processed only on the corresponding machine M_j. Each operation o_{ij} takes the corresponding processing time $\tau_{ij} \in \mathbb{N}$. All jobs from the set J need to be scheduled properly on the given machines, while trying to minimize the amount of spent time resources. We consider the following JSSP variation: each machine may process only one operation at the same time, operations related to one job can not be processed concurrently and processing of an operation can not be interrupted.

There are several types of target objective which can be used in evolutionary computations for the JSSP. We minimize the total flow-time of the schedule S:

$$F(S) = \sum_{i=1}^{n} (S(o_{\max_i}) + \tau_{o_{\max_i}}), \tag{5}$$

where $o_{\max_i}$ is the operation of the job J_i with the maximum start time in the schedule S, $S(o_{\max_i})$ defines the start time value of operation $o_{\max_i}$ and $\tau_{o_{\max_i}}$ is the processing time of operation $o_{\max_i}$.

4.1 JSSP Meta-Features

The JSSP meta-features were developed similarly to the TSP meta-features from Sect. 3.1. The following features present the JSSP instance properties:

1. M_{num} – the number of machines.
2. J_{num} – the number of jobs.
3. $MJ_{\text{ratio}} = \frac{M_{\text{num}}}{J_{\text{num}}}$ – the ratio between the number of machines and the number of jobs.
4. $\tau_{\min}$ – the minimum operation processing time.
5. $\tau_{\max}$ – the maximum operation processing time.
6. τ_{mean} – the mean operation processing time.
7. Dev_τ – the standard deviation of the operation processing time.
8. $\tau_{\text{avg}_M} = \frac{\sum_{j=1}^{m} \frac{\sum_{i=1}^{n} o_{ij}}{n}}{m}$ – the average processing time, which is also averaged per machine.

4.2 JSSP Auxiliary Objectives Generation

The restriction mentioned in the Sect. 3.2 (an auxiliary objective should have some property defined by the problem instance) is reached by the *The Shortest*

Job First (SJF) auxiliary objective generating method proposed by Lochtefeld et al. in [16].

Each particular job J_i has a minimum processing time, which can be calculated by the following equation: $F_{\min}(J_i) = \sum_{j=1}^{m} o_{ij}$. The next step is to define the subset of jobs H_k, which would be used in the k-th auxiliary objective. Then, the Eq. (5) evaluates the value of each auxiliary objective. The following algorithm is used to determine a subset of jobs H_k of the particular auxiliary objective:

1. The minimum processing times of all jobs $F_{\min_i}, i = 1 \ldots n$ are calculated.
2. The set of all jobs J is sorted with respect to the minimum processing time:

$$\text{Sort}(J) \rightarrow S_J = J_1, J_2, \ldots, J_n, \text{where} F_{\min}(J_i) \leq F_{\min}(J_j). \tag{6}$$

3. The sorted set of all jobs S_J is divided into r subsets with equal number of elements. Such a subset defines the auxiliary objective.

More details about the SJF auxiliary objectives can be found at [17].

The subsets H_k for the objectives can also be formed in a random way. Such a technique is used by Jensen in [4]. We test performance of the random composed subsets of jobs and the subsets generated by the SJF algorithm.

4.3 Experimental Evaluation on JSSP

In [4], Jensen also suggests using random auxiliary objectives for the JSSP problem. In [18] Petrova et al. apply the MOEA + RL method to this problem. We also consider the problem specific approach proposed by Lochtefeld et al. [16], based on job prioritization for auxiliary objective selection order. The OLHP method was compared with the aforementioned algorithms.

The JSSP instances were taken from the *Beasley's OR Library*[6]. The Generalized Order Crossover (GOX) [19] and the Position Based Mutation (PBM) were used in all the considered algorithms. EA solution candidates were represented as an ordered permutation list of different operations with repetitions. For example, an individual for a 2×3 problem may be encoded as $(1, 2, 2, 1, 1, 2)$, where the first "1" is the first operation of the job J_1, the second "1" is the second operation of J_1 and so forth. Furthermore, the Giffler-Thompson schedule builder [20] is used to transform genome to the proper solution candidate.

The crossover probability was set to 80%. The population size was $P_{\text{size}} = 100$. The stopping condition was whether an EA reached the limit of iterations. This limit was calculated as follows: $\text{iter}_{\max}(N, M) = N * M * 2$, where N was the total number of machines, M was the total number of jobs in the problem instance. Likewise in the TSP experiments, we used NSGAII algorithm. The number of EA runs for each training instance was $n = 4$. For the OLHP, Lochtefeld's, MOEA+RL methods we used 4 different SJF auxiliary objectives (remember that only one auxiliary objective was optimized simultaneously with

[6] http://people.brunel.ac.uk/~mastjjb/jeb/.

Table 2. Mean and standard deviation of resulting fitness (JSSP)

Problem	Best	OLHP	MOEA+RL	Lochtefeld	Jensen
abz6	7808	**8180.2 ± 143.8**	8261.3 ± 137.6	8213.2 ± 134.3	8244.6 ± 125.9
abz7	12561	**13116.1 ± 167.3**	13272.3 ± 180.7	13193.5 ± 188.9	13150.7 ± 179.7
abz9	12813	**13441.6 ± 187.7**	13574.6 ± 172.9	13475.9 ± 191.0	13463.0 ± 216.9
ft06	265	272.7 ± 7.0	272.0 ± 7.1	272.2 ± 7.0	**270.4 ± 6.8**
ft20	14279	**16047.1 ± 516.4**	16892.9 ± 542.8	16370.6 ± 499.5	16426.7 ± 590.5
la01	4832	**4989.6 ± 81.7**	5043.2 ± 96.6	5015.2 ± 87.6	5003.3 ± 71.3
la03	4175	**4236.4 ± 68.2**	4328.6 ± 70.5	4271.8 ± 64.6	4280.6 ± 73.9
la05	4094	**4189.0 ± 59.5**	4270.0 ± 61.6	4215.3 ± 63.5	4263.4 ± 66.1
la06	8694	**9298.3 ± 162.3**	9693.7 ± 201.4	9442.8 ± 215.1	9435.1 ± 189.1
la08	8176	**8708.9 ± 202.6**	9175.4 ± 194.6	8929.1 ± 221.8	8947.6 ± 229.2
la09	9452	**9777.7 ± 140.6**	10166.5 ± 170.2	9932.6 ± 170.4	9952.9 ± 211.5
la10	9230	**9557.7 ± 161.7**	10015.0 ± 214.6	9698.0 ± 176.8	9762.9 ± 189.8
la11	14801	**15840.5 ± 321.2**	16786.1 ± 367.3	16197.8 ± 339.1	16253.5 ± 434.1
la12	12484	**13449.3 ± 263.5**	14319.7 ± 301.4	13783.4 ± 339.3	13928.9 ± 377.5
la14	15595	**16199.8 ± 268.9**	17119.5 ± 300.9	16568.6 ± 338.3	16624.7 ± 341.3
la16	7393	**7836.2 ± 136.7**	7984.5 ± 151.4	7893.9 ± 158.1	7872.4 ± 145.4
la17	6555	**6847.5 ± 99.5**	6907.5 ± 96.3	6850.9 ± 88.1	6866.2 ± 99.5
la20	7427	**7751.9 ± 117.6**	7832.4 ± 144.0	7794.4 ± 124.8	7773.9 ± 119.5
la21	12953	**13940.4 ± 203.1**	14272.6 ± 200.6	14083.7 ± 253.4	14068.1 ± 219.9
la22	12106	13120.0 ± 221.2	13251.0 ± 213.5	**13094.6 ± 208.3**	13132.4 ± 223.9
la25	12465	**13154.3 ± 222.3**	13445.0 ± 243.1	13344.7 ± 260.6	13246.0 ± 214.8
la26	20234	**22351.7 ± 309.0**	22823.0 ± 312.0	22467.0 ± 272.3	22449.4 ± 311.8
la29	20404	**21498.7 ± 394.6**	22047.7 ± 386.5	21707.8 ± 383.1	21733.0 ± 392.2
la30	22333	**23725.8 ± 472.4**	24323.1 ± 382.8	24026.2 ± 471.2	23948.2 ± 419.7
la31	39007	**44400.9 ± 619.9**	45201.8 ± 541.6	44548.8 ± 580.2	44524.4 ± 614.9
la35	44059	**46014.8 ± 638.6**	46843.0 ± 633.2	46382.7 ± 679.1	46161.3 ± 784.9
la36	17073	**18461.3 ± 289.8**	18565.4 ± 244.1	18485.6 ± 272.2	18484.2 ± 270.3
la38	16621	**17346.9 ± 290.6**	17595.4 ± 280.3	17474.5 ± 280.0	17439.8 ± 282.9
la40	16618	**17667.4 ± 264.3**	17894.0 ± 270.7	17771.0 ± 267.2	17726.0 ± 274.6
orb02	7353	**7684.6 ± 123.8**	7753.5 ± 118.9	7739.5 ± 125.5	7708.2 ± 125.8
orb03	8280	**8772.8 ± 170.3**	8895.3 ± 194.0	8774.7 ± 189.7	8784.9 ± 235.5
orb06	8418	**8950.3 ± 205.2**	9113.8 ± 221.3	8979.7 ± 202.7	8950.6 ± 203.8
orb07	3296	**3505.6 ± 61.5**	3551.1 ± 74.9	3523.7 ± 63.2	3510.5 ± 74.1
orb09	7582	**8025.0 ± 180.5**	8231.8 ± 233.7	8062.8 ± 162.9	8149.8 ± 198.5
orb10	8043	**8335.7 ± 125.8**	8419.0 ± 146.9	8358.3 ± 147.4	8367.5 ± 134.5
swv01	20688	**24859.5 ± 711.2**	25710.4 ± 649.8	25441.2 ± 630.2	25461.3 ± 709.7
swv03	23266	**24617.6 ± 623.1**	25547.2 ± 633.0	25023.4 ± 668.7	25150.6 ± 652.5
swv04	24271	**25665.5 ± 574.4**	26457.5 ± 543.8	25978.6 ± 642.5	25957.5 ± 660.7
swv07	27385	**32738.5 ± 710.4**	33407.3 ± 652.0	32744.2 ± 711.8	32843.7 ± 755.0
swv08	32976	**36043.0 ± 775.3**	36655.9 ± 765.7	36136.3 ± 852.7	36265.8 ± 724.1
swv09	31841	**33783.7 ± 696.4**	34350.2 ± 744.4	34037.0 ± 855.6	33920.3 ± 788.5
swv11	108842	**140735.9 ± 2939.1**	145240.3 ± 3350.8	142351.2 ± 3360.4	141638.6 ± 3386.9
swv12	109128	**140695.3 ± 3173.4**	145495.7 ± 3093.0	142674.0 ± 2961.3	141281.9 ± 3247.0
swv14	126333	**137137.8 ± 3101.6**	141119.4 ± 3524.9	137850.9 ± 2779.5	137362.5 ± 3225.5
swv15	131037	**139467.0 ± 3991.3**	143849.0 ± 3676.6	140435.0 ± 3224.6	140211.6 ± 3476.1
swv16	113398	**117369.9 ± 1303.0**	119494.0 ± 1008.8	117937.3 ± 1045.1	117466.7 ± 1194.2
swv17	110145	**113689.1 ± 1020.0**	115666.0 ± 1042.6	114240.4 ± 981.0	113760.8 ± 1444.7
swv20	109742	**112866.7 ± 1060.6**	115285.1 ± 977.5	113277.0 ± 1038.2	113184.2 ± 1043.8
yn1	17317	**18199.5 ± 234.9**	18397.8 ± 212.6	18236.6 ± 210.0	18229.1 ± 227.3
yn4	19107	**19916.3 ± 253.7**	20115.0 ± 220.2	19997.4 ± 244.7	19959.7 ± 222.8
Total best		**48**	0	1	1

the target objective). Jensen's method was compared with the approach based on random generating of auxiliary objectives. We used the same OLHP implementation and the same frameworks as for the TSP.

Meta-Classifier Learning. The meta-classifier for selection of auxiliary objectives was trained on 32 JSSP instances (see Appendix A). The considered training instances were not so diverse as the training instances for the TSP. So we could cross-validate the learned classifier model on the training set of JSSP instances with high probability not to have a situation, when we would try to validate on some data, which was not considered in the learning process. The performance measure values of our classification model on JSSP after the 10-fold cross-validation were: Estimated Error Rate = 0.21, Precision = 0.72, Recall = 0.74, F-measure = 0.73. The correlation between the EA state features and selection of the most efficient auxiliary objective also exists for this benchmark problem.

Results of Solving the JSSP. There were 50 various JSSP test instances. Each instance was solved $\eta = 100$ times with NSGAII and each auxiliary objective selection method. We present comparison results in the same way as for the TSP problem. The averaged results for each method of objective selection with the standard deviations are listed in Table 2. The calculated comparison results show that the OLHP method has significantly outperformed all the other methods of auxiliary objective selection. It is also worth mentioning that we were not able to find sources with the best known solutions for the *Beasley's OR Library* problems. So we run a single objective EA 1000 times on each instance and gathered the best found results.

Statistical Testing. As in the TSP problem experiments, we used the Wilcoxon signed-rank test and the Holm-Bonferroni correction method for statistical verification of the results. We obtained the following adjusted p-values for pairs of objective selection methods: OLHP – MOEA+RL = 2.3e-09, OLHP – Lochtefeld = 2.3e-09, OLHP – Jensen = 2.3e-09. The aforementioned adjusted p-values show that average performance of the OLHP method and the other considered algorithms was significantly different. Moreover, the confidence level of this fact is more than 99%.

5 Conclusion and Future Work

The new method for selection of the most efficient auxiliary objective named *The Offline Learned Helper Picker* is proposed in this paper. The OLHP approach consists of two stages. At first, training instances of an optimization problem are used to build a meta-classifier for selection of auxiliary objectives. Properties of a problem instance and features derived from the fitness landscape of the current EA population compose the state of the evolutionary algorithm, i.e. the data vector for machine learning. Further, the trained meta-classifier is used to predict the most efficient auxiliary objective at different EA runtime points. Specifically, the selected auxiliary objective is predicted and optimized simultaneously with the target objective during a number of EA iterations.

The OLHP method was compared with similar approaches of objective selection on two NP-hard combinatorial problems: The Traveling Salesman Problem and The Job Shop Scheduling Problem. The newly proposed method outperformed the considered algorithms. Statistical significance of the obtained results was confirmed by the Wilcoxon signed-rank test followed by the Holm-Bonferroni correction.

In the future work we plan to use additional well-known statistical, probabilistic and informational measures for fitness landscapes. This should increase performance of the meta-classifier used for selection of auxiliary objectives. It is also desirable to use more computational power for obtaining experimental evaluation on real world problems. Better computational performance will also provide an opportunity to automatically find and use the most efficient parameters for EA with OLHP and other auxiliary objective selection methods. For example, the most efficient algorithm settings may be found with tools such as the irace package [21].

A Appendix: TSP and JSSP Instances Lists

TSP Train: att532, bays29, brazil58, ch130, d198, d493, eil101, gil262, gr120, gr202, gr24, gr431, hk48, kroA150, kroB200, kroE100, p654, pa561, pr136, pr264, rat575, rat99, si175, st70, ts225, u574, ulysses22. **TSP Cross-validate:** a280, att48, bayg29, bays29, berlin52, bier127, brazil58, brg180, burma14, ch130, ch150, d198, d493, dantzig42, eil101, eil51, eil76, fl417, fri26, gil262, gr17, gr21, gr24, gr48, gr96, gr120, gr137, gr202, gr229, gr431, hk48, kroA100, kroA150, kroA200, kroB100, kroB150, kroB200, kroC100, kroD100, kroE100, lin105, lin318, pcb442, pr76, pr107, pr124, pr136, pr144, pr152, pr226, pr264, pr299, pr439, rat195, rat99, rd100, rd400, si175, st70, swiss42, ts225, tsp225, u159, ulysses16, ulysses22.

JSSP Train and Cross-validate: abz5, abz8, ft10, la02, la04, la07, la13, la15, la18, la19, la23, la24, la27, la28, la32, la33, la34, la37, la39, orb01, orb04, orb05, orb08, swv02, swv05, swv06, swv10, swv13, swv18, swv19, yn2, yn3.

References

1. Pitzer, E., Affenzeller, M.: A comprehensive survey on fitness landscape analysis. In: Fodor, J., Klempous, R., Araujo, C.P.S. (eds.) Recent Adv. Intell. Eng. Syst., pp. 161–191. Springer, Heidelberg (2012)
2. Picek, S., Jakobovic, D.: From fitness landscape to crossover operator choice. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 815–822. ACM (2014)
3. Ceberio, J., Calvo, B., Mendiburu, A., Lozano, J.A.: Multi-objectivising the quadratic assignment problem by means of an elementary landscape decomposition. In: Puerta, J.M., Gámez, J.A., Dorronsoro, B., Barrenechea, E., Troncoso, A., Baruque, B., Galar, M. (eds.) CAEPIA 2015. LNCS (LNAI), vol. 9422, pp. 289–300. Springer, Heidelberg (2015). doi:10.1007/978-3-319-24598-0_26
4. Jensen, T.: Helper-objectives: using multi-objective evolutionary algorithms for single-objective optimisation. J. Math. Model. Algorithms **3**(4), 323–347 (2005)

5. Buzdalova, A., Buzdalov, M.: Increasing efficiency of evolutionary algorithms by choosing between auxiliary fitness functions with reinforcement learning. In: 2012 11th International Conference on Machine Learning and Applications (ICMLA), vol. 1, pp. 150–155. IEEE (2012)
6. Knowles, J.D., Watson, R.A., Corne, D.W.: Reducing local optima in single-objective problems by multi-objectivization. In: Zitzler, E., Thiele, L., Deb, K., Coello Coello, C.A., Corne, D. (eds.) EMO 2001. LNCS, vol. 1993, pp. 269–283. Springer, Heidelberg (2001). doi:10.1007/3-540-44719-9_19
7. Petrova, I., Buzdalova, A., Buzdalov, M.: Improved selection of auxiliary objectives using reinforcement learning in non-stationary environment. In: 2014 13th International Conference on Machine Learning and Applications (ICMLA), pp. 580–583. IEEE (2014)
8. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton (2011)
9. Kanda, J., Carvalho, A., Hruschka, E., Soares, C.: Using meta-learning to classify traveling salesman problems. In: 2010 Eleventh Brazilian Symposium on Neural Networks, pp. 73–78. IEEE (2010)
10. Kanda, J.Y., de Carvalho, A.C., Hruschka, E.R., Soares, C.: Using meta-learning to recommend meta-heuristics for the traveling salesman problem. In: 2011 10th International Conference on Machine Learning and Applications and Workshops (ICMLA), pp. 346–351. IEEE (2011)
11. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theory **13**(1), 21–27 (1967)
12. Jähne, M., Li, X., Branke, J.: Evolutionary algorithms and multi-objectivization for the travelling salesman problem. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pp. 595–602. ACM (2009)
13. Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: a case in study local optimization. Local Search Comb. Optim. **1**, 215–310 (1997)
14. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
15. Derrac, J., García, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol. Comput. **1**(1), 3–18 (2011)
16. Lochtefeld, D.F., Ciarallo, F.W.: Deterministic helper-objective sequences applied to job-shop scheduling. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 431–438. ACM (2010)
17. Lochtefeld, D.F., Ciarallo, W.: Helper-objective optimization strategies for the job-shop scheduling problem. Appl. Soft Comput. **11**(6), 4161–4174 (2011)
18. Petrova, I., Buzdalova, A., Buzdalov, M.: Improved helper-objective optimization strategy for job-shop scheduling problem. In: 2013 12th International Conference on Machine Learning and Applications (ICMLA), pp. 374–377, vol 2. IEEE (2013)
19. Bierwirth, C.: A generalized permutation approach to job shop scheduling with genetic algorithms. Oper. -Res. -Spektrum **17**(2–3), 87–92 (1995)
20. Giffler, B., Thompson, G.L.: Algorithms for solving production-scheduling problems. Oper. Res. **8**(4), 487–503 (1960)
21. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Technical report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)