LIII Simpósio Brasileiro de Pesquisa Operacional
João Pessoa - PB, 3 a 5 de novembro de 2021

AIISR - AI INSPIRED STOPPING RULE FOR GRASP METAHEURISTIC

Guilherme Caeiro de Mattos
PESC/COPPE/UFRJ
Av. Horacio Macedo, 2030, Cidade Universitaria, Rio de Janeiro - RJ, Brazil
mattosgc@cos.ufrj.br

Felipe Maia Galvão França
PESC/COPPE/UFRJ
Av. Horacio Macedo, 2030, Cidade Universitaria, Rio de Janeiro - RJ, Brazil
felipe@cos.ufrj.br

Luidi Gelabert Simonetti
PESC/COPPE/UFRJ
Av. Horacio Macedo, 2030, Cidade Universitaria, Rio de Janeiro - RJ, Brazil
luidi@cos.ufrj.br

Priscila Machado Vieira Lima
PESC/COPPE/UFRJ and NCE/UFRJ
Av. Horacio Macedo, 2030, Cidade Universitaria, Rio de Janeiro - RJ, Brazil
priscilamvl@cos.ufrj.br, priscila.lima@nce.ufrj.br

ABSTRACT

Metaheuristics are high level heuristics and usually lack an effective stopping criteria, relying on generic information, like number of iterations or time, to decide when they should stop. To tackle this problem for the GRASP metaheuristic, past works proposed a probabilistic stopping criteria based on the fact that the solutions of GRASP iterations follow a normal distribution. However, that method has two drawbacks: it is slow to estimate the probabilities and it is not very accurate. In order to address those problems, the present work proposes an alternative method that makes use of Decision Tree Regression to produce a stopping rule. In this paper, we show that this approach can be faster and more precise. We demonstrate that, in some cases, models trained on an instance of a given problem can also be applied to instances belonging to the same problem or even to other problems.

KEYWORDS. Stopping Rule. GRASP. Machine Learning Aided Optimization.

Paper topics: Computational Intelligence; Metaheuristics.

1. Introduction

In order to solve problems that are known to be NP-hard, that do not have a known optimal solution, metaheuristics, like Simulated Annealing, Ant Colony, Tabu Search and GRASP, are commonly employed. However, those techniques usually lack an effective stopping criteria and typically rely on generic information, like time or number of iterations without improvements to the best solution found, to decide whether they should stop searching for an improved solution. That can lead to one of two possible problems. The first is the case where the search stops too early and fails to find a better solution that was just a few iterations away, while the second is the situation where the global minimum is found, but the search continues because the algorithm has no way to recognize that, leading to an unnecessary consumption of resources.

Seeking to mitigate those problems in the GRASP metaheuristic, [Ribeiro et al., 2013] proposed a probabilistic stopping rule that, at any given iteration, is capable of estimating the probability of finding a minimum that is, at least, as good as the current best solution and uses that information to decide if the search should be stopped. To achieve that, the authors observed that solutions of GRASP iterations follow a normal distribution and used that knowledge to calculate their cumulative distribution function.

Despite being an effective method, the probabilistic stopping criteria also has its own drawbacks. Specifically, the calculation of the cumulative distribution function is slow and its results are not very accurate.

In this paper, we propose an alternative method based on machine learning, that tackles those problems by replacing the cumulative distribution function with a Decision Tree Regressor. In Section 2, a brief background on GRASP and the probabilistic stopping rule is provided, while in Section 3 the approach proposed by this work is discussed. The setup for the experiments conducted and the results are presented in Section 4. Finally, a conclusion is provided in Section 5.

2. Background

2.1. GRASP

GRASP, an acronym for *Greedy Randomized Adaptive Search Procedures*, is a multi-start metaheuristic initially proposed by [Feo, 1995] that is based in two steps, a construction phase and a local search, that are executed in sequence at every iteration, as shown in Algorithm 1. In the construction phase, the algorithm builds an initial solution and, to achieve that, it uses a greedy randomized search. That search procedure is similar to the usual greedy search, differing only in the way that the node expansion occurs.

While in a greedy search the edge followed is the one that returns the smallest cost (in a minimization problem), in a greedy randomized search a *Restricted Candidate List* (RCL) is created with the n best edges and one of them is randomly picked and followed. That random selection is what gives GRASP its *randomized* characteristic.

After that initial search is concluded, the solution obtained is then passed to a local search, that will investigate its vicinity, searching for a local minimum. Once it ends, the best solution is updated if the one returned by the local search is better than the current best, and the iteration is finished. In its basic form, a GRASP implementation requires the definition of only one or two parameters. One is n, that is the size of the RCL, and the other is the stopping criteria, that can be something as simple as a fixed number of iterations.

2.2. Probabilistic stopping rule

Proposed by [Ribeiro et al., 2013], the probabilistic stopping rule for GRASP is a method that takes advantage of the fact that the solutions obtained along GRASP iterations follow a normal distribution and calculates their cumulative distribution function (CDF). That function estimates,

Algorithm 1: Pseudocode for a simple GRASP implementation.

```
procedure grasp(maxIterations, rclSize)
1   bestSolution ← null
2   bestCost ← +∞
3   for i ← 1 to maxIterations do
4       solution ← greedyRandomizedSearch(rclSize)
5       solution ← localSearch(solution)
6       cost ← getCost(solution)
7       if cost < bestCost then
8           bestCost ← cost
9           bestSolution ← solution
        end
    end
10  return bestSolution, bestCost
end
```

for a given iteration, the probability of finding, in the next iteration, a minimum that is, at least, as good as the current best solution.

Formally, we have a random variable X, associated with the objective function value of the local minimum obtained at each GRASP iteration, and a probability distribution function $f_X(.)$ and a cumulative distribution function $F_X(.)$ of X. Then, let $f_1, ..., f_k$ be the set of solutions obtained in the first k iterations, and $f_X^k(.)$ and $F_X^k(.)$ be the estimates of $f_X(.)$ and $F_X(.)$, respectively, also in the first k iteration, the probability of finding, at iteration $k+1$, a minimum at least as good as the one found up to iteration k (defined as UB^k) is estimated by Equation (1).

$$F_X^k(UB^k) = \int_{-inf}^{UB^k} f_X^k(\tau)\, d\tau \tag{1}$$

Once that value is obtained, the stopping rule itself works by comparing it to a threshold β, provided by the user, that is a value between zero and one. If, at a given iteration, the estimate is grater than the β, the GRASP continues to be executed. If it is smaller than or equal to β, the execution stops. Further information about the method can be found in the original work.

3. AI-inspired stopping rule

3.1. Probabilistic stopping rule limitations

The stopping rule proposed by [Ribeiro et al., 2013] offers a way to mitigate two problems that affect standard GRASP implementations. One of them is early stopping, where the search for a solution stops too early. In some cases, if the execution could continue for one or a few more iterations, an improved solution was likely to be found.

The other problem, on the other hand, is the opposite of that and is directly tied to the fact that metaheuristics usually do not have a way to recognize if a given minimum is the optimal solution. In those situations, the global minimum is found, but the search continues, spending computational resources unnecessarily.

Despite its help with those things, the probabilistic stopping rule also has its own drawbacks. Most notably, it is slow to calculate and considerably deviates from the ”real”values it wants to estimate. This is demonstrated in Section 4.3.

3.2. *Learning* a stopping rule

In recent years, under the promise of providing tools that are, at least, as good - if not better - than humans in the tasks where they are applied, Artificial Intelligence has risen in popularity in the academia and in the business sectors. Among its subfields, the most prominent is *Machine Learning*, that studies techniques that learn from data.

In this work, we employ machine learning to the task of learning a stopping rule for GRASP. Specifically, we use a technique called Decision Tree Regression as a replacement for the cumulative distribution function used by [Ribeiro et al., 2013] in their method. It was chosen mainly because it is faster to train than many other popular non-linear estimators.

As for the training, it relies on data generated from previous GRASP executions. It uses the cost obtained at each iteration and other data derived from it as training features - the inputs -, and the number of minimums expected in the next iterations as a target, a value to be estimated. That value can later be converted into a probability.

All the data used to train the Decision Tree is obtained by executing GRASP a number of times and collecting data from each iteration. The number of iterations per execution should also be sufficiently large, in order to allow the target variable (*remaining_mins*) to be as precise as possible. The details about the datasets used for training can be found in Section 4.2.

The approach proposed in this work also has some other characteristics that are important to mention. Specifically, by training a model solely on a single instance of a given problem, its usefulness is, at first glance, questionable. However, some tests showed that there is a level of compatibility between some instances. That can be seen in Figure 1, that is explored in Section 4.3.

That compatibility is the heart of the proposed approach. However, it is also what presents the biggest challenge, because, at the moment this paper is written, there is still no straightforward method to identify what is compatible with what, other than testing it in practice. That aspect will be revisited in Section 4.4.

Regardless of the challenges posed, the remaining component of the method - the β value - works exactly as in the probabilistic stopping rule of [Ribeiro et al., 2013]. It considers the existence of a threshold (β), provided by the user, that is recurrently compared to the probability estimated by the Decision Tree Regression model and, if the value produced by the regressor is lower than or equal to that threshold, the execution is stopped.

4. Experimental results

4.1. Experimental setup

The tests presented in this paper were executed on a series of different platforms. Model training and data manipulation ran on virtual machines in the services Google Colaboratory and Kaggle Notebooks. The first provides to each machine 1 CPU core (with two threads) of a Xeon processor and 12.69GB or RAM, while the second offers 2 CPU cores (with two threads per core) and 16GB of RAM. For both services, specific details, like storage type and processor models are unavailable. In Kaggle's case, the processor can be a Xeon, from Intel, or an Epyc, from AMD. In both services, the operating system is Ubuntu 18.04.5 LTS.

For time tests, the platform was a local machine that has an Intel Core i7 4720HQ with 4 cores and 8 threads, 16GB of DDR3 memory (running at 1600Mhz) and a SATA SSD BX500 (CT480BX500SSD1) from Crucial. The operating system is Windows 10, but the codes were executed within the Windows Subsystem for Linux (version 1), running Ubuntu 18.04.4 LTS.

4.2. Problems and datasets

In order to evaluate the approach proposed in Section 3 and compare it to the probabilistic stopping rule, this work selected two of the problems used by [Ribeiro et al., 2013] to evaluate their own technique. Those are the *Set k-Covering Problem* and the *Quadratic Assignment Problem*.

The Set k-Covering Problem is a variant of the Set Covering Problem (SCP). Given a set $I = \{1, ..., m\}$ and a set $P = \{P_1, ..., P_n\}$, composed of subsets of I, where each P_j is associated to a cost c_j, with $j = 1, ..., n$, the SCP aims to find a set J that covers I at a minimal cost. In other words, it seeks $J = I$, where J is the union of a number of subsets P_j and the sum of the costs c_j is minimal. The Set k-Covering Problem is a modification of the usual SCP problem, where each element of I should be covered by, at least, k elements of P.

Introduced by [Koopmans and Beckmann, 1957], the Quadratic Assignment Problem (QAP) originally tries to solve the problem of allocating facilities to places. In that problem, there is a set of n facilities and a set of n places and, for each pair of places, there is a distance and, for each pair of facilities, an associated flow. Given that set up, the QAP seeks to allocate facilities to places in such a way that the sum of the products between the distances and the flows is minimal.

With those problems selected, a set of their instances was chosen. Specifically, for the SCP, five benchmark instances from the OR-Library [Beasley, 1990] were picked. They are the *scp42*, the *scp47*, the *scp55*, the *scpa2* and the *scpb5*, and were introduced by [Balas and Ho, 1980] and [Beasley, 1987]. Among them, the first four were also employed by [Ribeiro et al., 2013], while the fifth was selected in order to include an additional larger instance. Their specific characteristics can be seen in Table 1. In the context of this article, all executions of the instances SCP where run with $k = 2$ (parameter of the set k-covering problem).

instance	m	n
scp42	200	1000
scp47	200	1000
scp55	200	2000
scpa2	300	3000
scpb5	300	3000

Tabela 1: Characteristics of the SCP instances.

As for the ones chosen to represent the QAP problem, they were the *tai30a*, the *tai50a*, the *tai100a* and the *tai100b*. They were introduced by [Taillard, 1991] and [Taillard, 1995] and were obtained from the QAPLIB [Burkard and Rendl, 1997]. The two not present in [Ribeiro et al., 2013] were the *tai100a* and the *tai100b* and were picked for the same reason the *scpb5* was. Table 2 presents their specific attributes.

instance	n
tai30a	30
tai50a	50
tai100a	100
tai100b	100

Tabela 2: Characteristics of the QAP instances.

After all the instances were obtained, they were employed to generate datasets to train and evaluate the Decision Tree Regressors. Each dataset was constituted by a total of 20 executions (generated with different seeds) of a single instance. In order to keep the methodology close to what was done by the creators of the probabilistic stopping rule, all executions ran for $1\,000\,000$ iterations.

That large value was necessary to increase the confidence on the values attributed to *remaining_mins* in the dataset. In total, for each instance, a dataset composed of $20\,000\,000$ iterations was generated.

For each iteration, a series of features was derived. Table 3 displays them alongside with their type and meaning. The target, in all datasets, is *remaining_mins*. For a given iteration, of a given execution, that value is the number of remaining costs, within that execution, that are smaller than the best_cost in that iteration.

Feature	**Type**	**Description**
iteration	integer	iteration number
cost	float	solution cost
best_cost	float	smallest cost up to the current iteration
best	boolean	flag telling if best_cost changed in the current iteration

Tabela 3: Features derived from an iteration.

For those features, the correlation matrix was calculated (using the pearson correlation coefficient) and their correlation to *remaining_mins* was observed. In Table 4, the averages of the correlation values across executions of all 9 instances considered in this work are presented. Each problem was represented by all their instances, and each instance by two executions. Based on it, it is possible to conclude that the features with the highest correlation to *remaining_mins* are *best_cost* and *best* and that these two have a very low correlation between themselves. The standard deviations of the correlations between those same two features and *remaining_mins* across the executions were 0.0944 and 0.0854, respectively. For those reasons, the combination of $\{$*best_cost, best*$\}$ was chosen to describe an iteration.

	iteration	cost	best_cost	best	remaining_mins
iteration	1	-0.0007	-0.6633	-0.0059	-0.0261
cost	-0.0007	1	0.0005	-0.0094	0.0008
best_cost	-0.6633	0.0005	1	0.0446	0.1879
best	-0.0059	-0.0094	0.0446	1	0.2518
remaining_mins	-0.0261	0.0008	0.1879	0.2518	1

Tabela 4: Average of the correlation between the features and *remaining_mins* across a series of different executions, from different instances and problems.

4.3. Results

Starting from the training of the Decision Tree Regressor, each dataset, belonging to each instance, was trained into a model, totalling 9 different models. In Table 5, the cross-validation scores for a 10-fold cross validation are shown under two different metrics. The *mean absolute error* (MAE) allows us to see that, on average, the error was very low, staying bellow 2 for most models, while the low *root mean squared error* (RMSE) indicates not only that, but also that, among the estimates, there were no major deviations from the target, because it would have caused that error to grow significantly. In all cases, the model was created using the Decision Tree Regressor available in the Python library Scikit Learn, and all parameters were kept to their default values, except for the maximum depth that a tree can reach, that was set to 10.

Those results, alone, are of little use. However, as written in Section 3, the main interest is to observe how models trained on a given instance perform when evaluated on datasets from other

instance	CV MAE	CV RSME	instance	CV MAE	CV RMSE
scp42	1.870	34.060	tai30a	1.994	41.188
scp47	1.409	34.704	tai50a	1.827	26.913
scp55	1.778	36.667	tai100a	1.239	43.699
scpa2	1.870	28.333	tai100b	2.136	32.880
scpb5	1.611	35.342			

Tabela 5: 10-fold cross-validation score for models trained with trees of depth up to 10.

instances of the same problem, or even from other problems. That behavior can be observed in Figure 1, where each line represents a model (identified by the labels to the left) and each column a dataset (with 20 executions) where that model was evaluated on (identified by the instance names at the bottom). In this case, instead of using *mean absolute error* or *root mean squared error*, the metric employed was a variant of the *symmetric mean absolute percentage error* (SMAPE), that is defined by Equation (2).

$$SMAPE = \frac{\sum_{t=1}^{n} |F_t - A_t|}{\sum_{t=1}^{n} (A_t + F_t)} \tag{2}$$

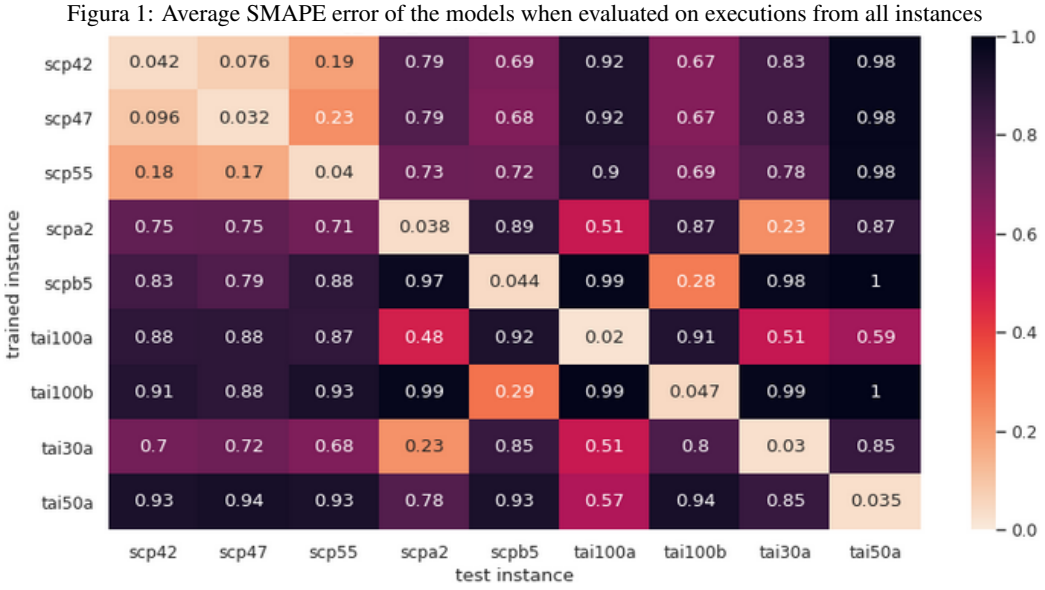
That variant was used because it gave a better overall sense of what was happening in each dataset that a model was evaluated on, if compared to the other two metrics. Another reason for its use was the fact that it is somewhat easier to interpret because, under ideal conditions (if the denominator is not equal to 0), it returns a percentage between zero and one. It is also important to note that the results are not the overall error for the entire datasets. To give a better overall sense of how the models were performaing at execution level, the numbers present in Figure 1 represent the average SMAPE per execution for that given dataset.

In that same Figure, it is possible to observe some interesting results. Firstly, the models from the instances *scp42*, *scp47* and *scp55* perform quite well on datasets belonging to each other, while the models for *scpa2* and *scpb5* do not, despite being from the same problem as the first three instances. However, surprisingly, those two instances have a good *compatibility* with some instances from QAP, that are completely different from the SCP instances. Another relevant finding is that those relations are reciprocal, so, if a model from instance x performs well on the dataset from an instance y, the model trained on y also performs well on the dataset from x.

Now, comparing the performance of our models with what is offered by the probabilistic stopping rule of [Ribeiro et al., 2013], there are substantial differences. The first, that is one of the problems that we sought to attack from the beginning, is related to time. In our tests, running their estimator for $1\,000\,000$ iterations, we observed that the time required to output the probability increased over time and, on average, it took 0.46 seconds on a sample *scp42* execution. The villain in this case was the time required to calculate the CDF.

In our approach, on the other hand, the time required to estimate the *remaining_mins* and calculate the probability was orders of magnitude smaller. However, that also came at a cost of training a model and, at each iteration, normalizing the data from that iteration (using a MinMaxScaler fitted on the evaluated dataset), just so that it can be fed to the regressor. The specific numbers can be seen in Table 6. It is also worth noting that, in our case, the time required to acquire the dataset needed for training is the main drawback. It varies from instance to instance, but, for most datasets used in this work, it took more than 24h to generate 20 executions, with $1\,000\,000$ iterations each.

Figura 1: Average SMAPE error of the models when evaluated on executions from all instances



technique	normalization (s/iter.)	training (s)	estimation (s/iter.)
Ours	0.0000012	15.011	0.0001702
[Ribeiro et al., 2013]	—	—	0.4642875

Tabela 6: Average times per phase of each technique, where, under *estimation*, a *scp42* execution is considered.

Lastly, we also compared how the two approaches perform on the stopping rule itself. In Table 7, we have a *scp55* execution being evaluated by a model trained on a *scp42* dataset and by the estimator of [Ribeiro et al., 2013]. The results show that our approach was more accurate in its predictions. That behavior is better seen in Figure 2, where the estimates of both techniques can be seen across two sections of the *scp55* execution.

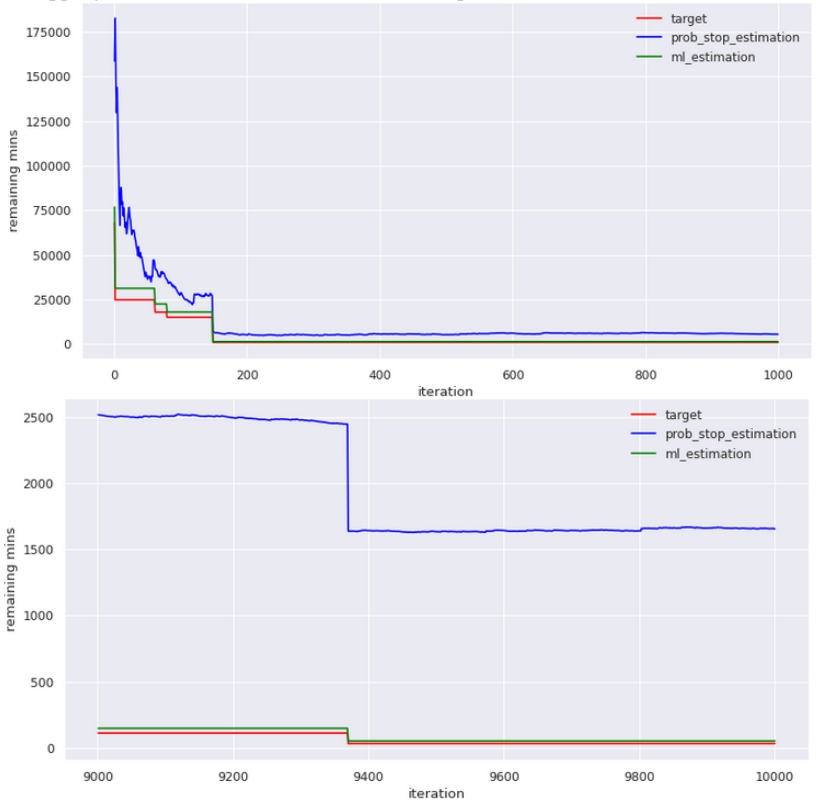
β	iteration	Ours	[Ribeiro et al., 2013]	real prob.	remaining mins.
0.100000	1	0.076794	—	0.067986	67986
0.010000	150	0.001427	0.006852	0.001032	1032
0.001000	1130	0.000780	0.004343	0.000528	528
0.000100	9371	0.000051	0.001637	0.000032	32
0.000010	172166	0.000003	0.000840	0.000005	5
0.000001	325068	0.000001	0.000513	0.00000	0

Tabela 7: Test of the stopping rule for a scp55 execution evaluated using an scp42 model and the probabilistic stopping rule of [Ribeiro et al., 2013].

4.4. Discussion

The results presented in Section 4.3 show that the approach proposed by this paper is promising, being able to be relatively faster than the original probabilistic stopping rule and more accurate. When it comes to speed, it is *relatively* faster because our method requires an initial

Figura 2: Comparison between the performances of the proposed approach, using a scp42 model, and the probabilistic stopping rule of [Ribeiro et al., 2013], on a scp55 execution.



training phase, that also has to account for the time required to acquire the dataset. However, once it is trained, it can be used repeatedly at a very small cost in time and computational resources. The time needed to obtain the datasets could also be neglected in situations where our approach is applied on already existing optimization processes with GRASP, that uses some generic stopping rule, where the costs obtained at each execution could be reused to build a dataset.

When it comes to accuracy, we also showed that, in our approach, a given model, of a given instance, might be able to be used on multiple different instances, even from different problems, but at varying performances. However, considering that our study is still at early stages, we are still investigating what are the reasons that lead to that possibility. In some cases, we might be able to argue that the similarities between the instances seem to allow that, because, if we compare the instances *scp42*, *scp47* and *scp55*, they are quite similar in their attributes. However, that assumption fails when we realize that the instances *scpa2* and *tai30a* are compatible, despite belonging to different problems and being completely different from each other.

5. Conclusion

In this paper, we presented a machine-learning based method as an alternative approach to the probabilistic stopping rule and showed that it has potential to overcome the limitations of the latter. Although not fully general as the probabilistic stopping rule, our method has the advantage of being faster to compute, and we have shown how to profit from using a thoroughly analyzed instance as a model for other instances. The experimental analysis pointed out the possibility of identifying some instances as models even for instances of other problems. Future work includes

experimenting with several machine learning paradigms as well as broadening the range of problems covered. Further research consists of better understanding the model selection process.

Acknowledgements

We would like to thank the authors of [Ribeiro et al., 2013] for providing the source codes of their estimator and of their GRASP implementation to support this work.

This study was financed in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Referências

Balas, E. e Ho, A. (1980). *Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study*, p. 37–60. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-00802-3. URL https://doi.org/10.1007/BFb0120886.

Beasley, J. E. (1987). An algorithm for set covering problem. *European Journal of Operational Research*, 31(1):85–93. ISSN 0377-2217. URL https://www.sciencedirect.com/science/article/pii/037722178790141X.

Beasley, J. E. (1990). Or-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 41(11):1069–1072. ISSN 01605682, 14769360. URL http://www.jstor.org/stable/2582903.

Burkard, S. E., R. E.and Karisch e Rendl, F. (1997). Qaplib – a quadratic assignment problem library. *Journal of Global Optimization*, 10(4):391–403. ISSN 1573-2916. URL https://doi.org/10.1023/A:1008293323270.

Feo, M. G. C., T. A.and Resende (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133. ISSN 1573-2916. URL https://doi.org/10.1007/BF01096763.

Koopmans, T. C. e Beckmann, M. J. (1957). Assignment problems and the location of economic activities. *Econometrica*, 25(1):53–76. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1907742.

Ribeiro, C. C., Rosseti, I., e Souza, R. C. (2013). Probabilistic stopping rules for grasp heuristics and extensions. *International Transactions in Operational Research*, 20(3):301–323. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12010.

Taillard, D., E (1995). Comparison of iterative searches for the quadratic assignment problem. *Location Science*, 3(2):87–105. ISSN 0966-8349. URL https://www.sciencedirect.com/science/article/pii/0966834995000086.

Taillard, E. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4):443–455. ISSN 0167-8191. URL https://www.sciencedirect.com/science/article/pii/S0167819105801474.