Algorithm Selection for Combinatorial Packing Problems

Ioana Sitaru*, Madalina Raschip†
Faculty of Computer Science, "Alexandru Ioan Cuza" University of Iasi
Iasi, Romania
Email: *ioanasitaru14@gmail.com, †madalina.raschip@uaic.ro

***Abstract*—This paper describes a way of building an algorithm selection system for the two-dimensional rectangle packing problem. A list of features was defined, and five heuristics and metaheuristics were selected as solvers. The first step involves the extraction of features from the input problem instances. Using those computed features, machine learning regressors will predict the performance of each solver on the instance and, therefore, find the most fitted algorithm for it. Both the predicted execution time and the expected quality of the solution were used as performance metrics for the selection step. The experimental results showed that the built system has a high accuracy in predicting the best algorithm based on the runtime execution and the solution quality. Moreover, the system achieves these results with much lower computational resources than running the actual solvers of the problem.**

***Index Terms*—automated algorithm selection, strip packing, runtime prediction, performance prediction**

I. Introduction

There is a wide class of packing problems as they involve many applications, especially in the manufacturing industry, that have been modelled and tackled as combinatorial optimization problems. Among the real-life applications, we mention the waste minimisation in wood, glass or textile industries [10], text or image placement on a sheet, warehouse loading or scheduling problems [1]. The objective in this class of problems is the packing of objects into containers. There are many variations of the problem, such as 2D packing, packing by weight, packing by cost, bin packing, etc. In this study, we considered the rectangle packing problem. More precisely, we tackled the two-dimensional strip packing problem: a set of rectangular items must be placed into a rectangular container such that some constraints are satisfied. The objective is the minimization of the waste. The problem was proved to be NP-hard [2].

Exact algorithms, like linear programming or branch and bound algorithms that can find the optimal solution, do not guarantee obtaining the solution within a reasonable amount of time. Especially when a large number of items are to be placed, the time required for a problem to be solved is unpractical. Therefore, heuristic and metaheuristic approaches were developed in order to find good approximations of the optimal solution. Baker et al. [2] studied the strip packing problem for the first time and proposed the Bottom-Left heuristic, which places new rectangles in the strip as low as possible and then as left as they fit. The Guillotine algorithm and its variations [3] yielded good results on the packing problem. Other heuristic approaches that have been proposed are the Bottom-Left-Fill algorithm [4] and the heuristic recursive algorithm [5]. The Best-Fit algorithm, which includes a step that computes the rectangle to be placed next based on the dimension that best fills the gaps, was described in [6]. A summary across several heuristics has been gathered by Jylänki [3] with applications to the rectangle bin packing problem. Different approaches based on metaheuristics have been proposed in the literature. One of the earliest works that tackled the problem by using a combination of heuristics and metaheuristics is [7]. The authors used a genetic algorithm to establish the optimal order of the rectangles to be placed. The genetic encoding represented each chromosome as a permutation and used a heuristic for evaluating the fitness of the solution. Another evolutionary approach that considered the number of independent cuts as an optimisation criterion has been proposed by Illich et al. in [8]. A survey of heuristics and metaheuristics methods for bin packing and strip packing problems is given in [9].

There has been great research on different variations of the problem and, consequently, many approaches to solving it have arisen, none outperforming the others for all inputs. The optimal algorithm choice is strongly dependent on the input data. This is no surprise as it has been long proven that no single algorithm will be the best option for all possible instances [10]. Given the No Free Lunch theorem and the fact that the most suitable algorithm is dependent on the instance it needs to solve, an approach based on algorithm selection would be appropriate. The algorithm selection problem has been addressed initially by Rice in [14]. Since then, the field has developed considerably and it has substantially improved the state of the art results in difficult combinatorial optimization problems like propositional satisfiability [17] and the traveling salesman problem [18]. An overview of the research in automated algorithm selection is given in [19].

In the context of the strip packing problem, although the problem has been long studied, only recent research has been conducted in the algorithm selection area. Rakotonirainy [16] has proposed a framework based on machine learning techniques for selecting metaheuristic algorithms for solving the strip packing problem. The approach uses classification as the selection procedure, as opposed to regression to predict the best performing algorithm.

In this paper, we propose an algorithm selection system

2022 IEEE Congress on Evolutionary Computation (CEC) | 978-1-6654-6708-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/CEC55065.2022.9870417

for the two-dimensional rectangle packing problem. Selecting the best algorithm on a per-instance basis implies using empirical performance models to predict the performance of all algorithms in the portfolio. Having these predictions, we can choose the algorithm predicted to perform best. We consider the quality of a run as a trade-off between the runtime of the algorithm and how close to the optimum the obtained solution is. Therefore, both metrics were used in building the algorithm selection system.

The paper is structured as follows. Section II reviews the strip packing problem. In section III we describe the algorithms used for the portfolio, while in section IV we present the system. In the experimental section V, we analyse the performance of the system and highlight the gains achieved. Section VI concludes the paper.

II. The Two-Dimensional Strip Packing Problem

The Two-Dimensional Strip Packing Problem (2SP) is defined next. The input consists of a set $I = \{1, 2, \ldots, n\}$ of n rectangles, each with a specific width w_i and height h_i, and the width of the strip W. The aim is to place all rectangles into the strip of the given width, having the objective to minimize the strip height H needed for the packing. The location of each rectangle i in the strip is represented by the coordinates (x_i, y_i) of its bottom left corner. The problem can be modelled as follows:

$$
\begin{array}{lll}
\min & H & \\
\text{s.t.} & x_i + w_i \leq W, & \forall i \in I \\
& y_i + h_i \leq H, & \forall i \in I \\
& x_i + w_i \leq x_j \ or \ x_j + w_j \leq x_i \ or & \\
& y_i + h_i \leq y_j \ or \ y_j + h_j \leq y_i, & \forall i, j \in I, i \neq j \\
& x_i, y_i \geq 0 & \forall i \in I
\end{array}
$$

Although the problem can be straightforwardly modelled in the linear programming or constraint programming frameworks, the number of constraints introduced grows with the number of rectangles to be placed, and even for the easiest instances, the sets of constraints introduce a lot of slack variables. Although the problem description is simple, an optimal arrangement of the rectangles seems hard to find. The 2SP problem is NP-hard as it can be reduced to the one-dimensional bin packing problem [2].

III. Problem solvers

In this section, heuristic and metaheuristic approaches used for solving the Two-Dimensional Rectangle Strip packing problem, which are part of the portfolio, are described. The heuristic algorithms for strip packing can be divided into constructive heuristics and improvement heuristics. Constructive heuristics add rectangles in an incremental way, while for the second category, the initial solution is improved by applying some small changes. For portfolio construction, we have considered algorithms from both categories.

A. The binary tree approach

The binary tree algorithm, also mentioned in the literature as the Guillotine algorithm [3], uses a binary tree data structure to keep track of the remaining strips. Given that the strip size is not priorly known, the algorithm starts with the smallest height that can accommodate the first rectangle.

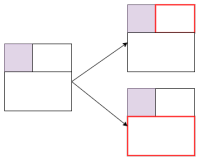


Fig. 1. Problem split after rectangle placement

The placement of any rectangle divides the original problem into two smaller ones, each being separately handled, as Figure 1 describes. The second rectangle to be placed will first try to fit to the right of the already placed rectangle. In this way, the lines are first filled if possible, so that the height does not grow. The procedure grows down the strip only if the rectangle does not fit there, as the width is not large enough.

B. Priority heuristic

The priority heuristic [11] consists of a recursive process which works by assigning to each unplaced rectangle a priority. The priority assignment is done based on five placement scenarios depicted in Figure 2.

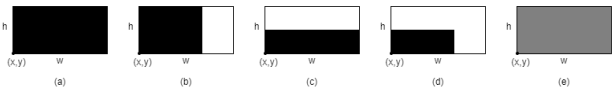


Fig. 2. Priority assignment based on placement scenarios

The rectangle placement captured in Figure 2 (a) gets the highest priority 1, because it is the ideal case in which the rectangle fits perfectly into the remaining strip. The rectangle in (b) gets priority 2 as it fits the already used height of the strip, while for the width fit (c), we assign priority 3. In case (d), the rectangle fits by splitting the strip into two smaller strips, so the rectangle gets priority 4. The lowest priority is assigned to rectangles that do not fit the strip at all, meaning they cannot be placed in the given sub-strip (e).

After the assignment of priorities for all unplaced rectangles, the one with the highest priority will be placed, and the remaining strip is computed. If multiple unplaced rectangles have the same priority assigned, the first one in the list is placed. This process is repeated recursively until the list of unplaced rectangles is empty.

For rectangle placements that fit as in Figure 2 (d), the remaining space needs to be divided into two strips. There are

two possible ways. The decision is taken by looking at the smallest side of all the remaining rectangles in comparison to the already placed rectangle sides.

C. Area based heuristic

Area based heuristics [3] have been used for packing problems with the aim of minimizing the enclosing object. Our approach chooses the rectangle to be placed next based on the area that it leaves unpacked after the item placement. The idea behind it is that, for each placement, it tries to minimize the leftover area so that the packing is as tight as possible. It grows the height of the strip only when no item can be placed on the strip. After the best rectangle to be placed is found, i.e. the one that leaves the minimal area of the strip, the strip is divided and this process is executed recursively until all items are packed.

D. Greedy approach

To have an upper bound performance, we implemented a greedy approach. It was designed based on the First-Fit Decreasing algorithm studied on the bin packing variant of the problem [24]. Our method uses a grid of fixed width W as the space of all possible placements of the items. The algorithm iterates this space for every rectangle that needs to be placed. The chosen solution for an item will be the first placement where it fits, meaning that it satisfies the constraint that rectangles should not overlap each other.

This approach yields a solution containing the placement coordinates for all rectangles and the height of the needed strip to fit all items. The objective is to minimize the strip height, meaning that the placed rectangles should be as tight as possible. For that, the traversal of the grid is done from the bottom to the top using a line search. In this way, the height will not grow unless the rectangle does not fit into the already assigned strip.

E. Genetic approach

The approach considered here is similar to Jakobs' seminal work [12] and it was inspired by previous experiments that demonstrated that the order in which a deterministic approach considers the rectangles has a significant influence on the final solution. The genetic algorithm is used to minimize the height of the strip, and a deterministic algorithm, with a solution close to the optimal one, is used for rectangle placing.

In this case, the individual is a permutation of rectangles. The fitness function is the strip height obtained by using the permutation of rectangles as input to a deterministic algorithm. The mutation and crossover operators are the classic ones used for permutation representations [13]. We experimented with order crossover and shuffle index mutation.

For the placement, we have used the priority heuristic [11], detailed in section III-B. As already mentioned, the heuristic picks up the rectangle that has the highest priority for placement, but if multiple rectangles have the same priority, the first one in the list is picked up. In this quite common situation, the order established by the genetic algorithm step comes into play.

The fitness of the individual is the obtained value for the strip height. In this way, the individuals in the population are ranked by their quality of solution, and the goal is to minimize the objective function.

IV. Algorithm selection

Under the hypothesis that no algorithm is the best over all instance types, an empirical performance model will be constructed. This model will be used to predict the performance of all candidate algorithms and select the one predicted to perform the best.

An algorithm selection system consists of four components: the problem space P (section II) , the feature space F (subsection IV-A), the algorithm space A (subsection IV-B) and the performance measure space W (subsection V-B). The components interact based on Rice's model [14] as presented in Figure 3. For each train instance, the features defined in the feature space are computed. Having the algorithm portfolio, the training instances are run on each algorithm and the performance measure is recorded. The algorithm performance measure is combined with the instance characteristics in order to train a performance predictor for each algorithm. This will be used in further inferences of the system to predict each algorithm's behaviour on the given new data.

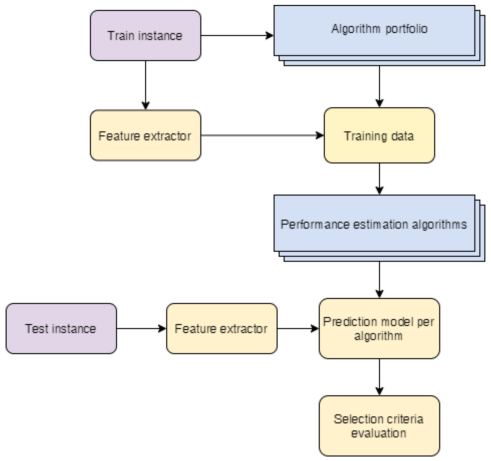


Fig. 3. Algorithm selection system

A. Feature space

In order to find a mapping between the input data and the performance of each candidate algorithm, we will extract some attributes that characterize the instances of our problem.

The strip width and the rectangles list, i.e. a list of pairs *(width, height)* for each item will be used as input data for the strip packing problem. From these, we extracted a set of 23 features that characterize the input instance. The features are described in Table I. Some of the features have been previously used in the literature for variants of the packing problem [15], [16]. Features are statistics of the width, height, items, and items areas. Inspired by applications of algorithm selection to

other optimization problems, we also added a basic solver's result as a feature. The *baseline strip height* feature implies solving the packing using a shelf division [25] to get an upper limit for the solution that can be obtained. This solver has a very fast execution time, allowing us to compute all 23 features almost instantaneously for a new instance.

TABLE I
THE FEATURES USED

No.	Feature name	Description
1	max items area	The maximum area from all items
2	total items area	The sum of all items areas
3	mean items area	The mean of all items areas
4	median items area	The median of all items areas
5	variance items area	The variance of all items areas
6	mean height	The mean height over all items
7	mean width	The mean width over all items
8	variance height	The variance height over all items
9	variance width	The variance width over all items
10	ratio smallest-biggest	The ratio between the smallest and the biggest item
11	square items	The number of items that have an almost square format
12	non-square items	The number of items that have a large/small height-width ratio
13	height ratio	The ratio between min height and max height
14	width ratio	The ratio between min width and max width
15	no items	The number of items
16	strip width	The input strip width value
17	max aspect ratio	The max ratio between largest side and smallest side
18	max area ratio	The max difference in area between any two items
19	heterogeneity ratio	The number of different item types divided by total number of items
20	width ratio	The ratio between strip width W and mean value of items' width
21	items width ratio	The sum of items largest side divided by strip width W
22	baseline algorithm strip height	The strip height needed for a baseline algorithm for packing the items
23	baseline algorithm runtime	The runtime needed for the baseline algorithm

B. Algorithm space

The algorithm portfolio consists of all solvers that candidate to be the best choice for a given instance. For building a robust algorithm selection system, the algorithms should all solve the given problem, but none of them outperforming the others for all types of instances. For the rectangle strip packing algorithm portfolio, we used the algorithms presented in section III. Therefore, our algorithm portfolio consists of five candidates: the greedy approach, the genetic algorithm, the binary tree heuristic, the priority heuristic, and the area heuristic.

V. EXPERIMENTAL RESULTS

A. Datasets

The basis of any machine learning system is the data that it gets trained on. This is a critical piece of the ensemble, and it has a huge impact on the overall results. The training data needs to be as diverse as possible.

There is not a lot of open-source data available for the strip packing problem, but we managed to gather a set of benchmark instances taken from the literature published over the years. The datasets used are described in Table II.

TABLE II
THE DATASETS USED

data source	instances	number of rectangles	strip width
Bengtsson [20]	10	20-200	25-40
Berkely & Wang [21]	300	20-100	10-100
Martello & Vigo [22]	200	20-100	10-100

The data needed preprocessing and storing in a uniform format over all instances. A total of 510 instances have been used for running the implemented solvers on.

B. Selection criteria

We used two criteria for selecting the best algorithm. The first metric considered is the runtime of the algorithm. We need to build a prediction model that is capable of estimating the runtime given a set of instance features. In addition, the proposed algorithms achieve all near optimal solutions, and the results are not equally good in between the approaches. Consequently, we added the resulted strip size as another criterion for cases when we are interested in finding the best solution quality.

C. Experimental setup

Building a prediction model requires beforehand training data. For this, we ran the algorithms from the portfolio on all instances until a solution was found, except for the genetic algorithm. The parameters of the genetic algorithm were set empirically and we recorded an average over 20 runs for the two metrics for the genetic algorithm, each run computing 100 of generations. After obtained a packing solution from a solver, we stored the result, namely the strip height needed to pack the items, and the runtime of the algorithm. In this way, we constructed the target values for our predictors. We used the extracted features described in Section IV-A as instances characteristics. As a result, we constructed the datasets for both the runtime and the solution quality prediction tasks. We split the instances into 75% for training and 25% for testing.

An evaluation procedure of the models must be established. We used the root mean square error (RMSE) for prediction error measurements. RMSE is computed as the root square of the differences between predicted values and observed values in relation to the number of observations.

$$RMSE = \sqrt{\Sigma_{i=1}^{n} \frac{(\hat{y_i} - y_i)^2}{n}}$$

The larger the RMSE value, the worse the predictions of the model are. The comparison of the RMSE values across different models allowed us to determine the best model from the cross validation phase.

D. Runtime prediction

1) Regression models: For the runtime prediction, we need to construct a regression model for each algorithm. These models should be capable of predicting the runtime of an algorithm to solve an instance based on the extracted instance characteristics.

We investigated the performance of multiple regression models on the task of predicting the runtime based on computed instance features. We considered for evaluation the following regressors: linear regression (LinReg), Automatic Relevance Determination regression (ARD), extreme gradient boosting (XGB), support vector regression (SVR), k-nearest neighbours (KNN), decision tree (DTR), and random forest (RFR). For each of them, we ran cross validation on the dataset and recorded the mean RMSE obtained. The comparison results are shown in Figure 4.

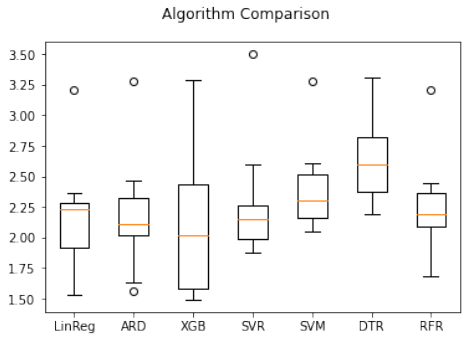


Fig. 4. Regression models comparison for runtime prediction

The experimental results show that XGBoost is performing the best for our task, especially with additional hyper-parameter tuning. To find the optimal parameter combination that best describes our dataset, we applied the grid search technique which allowed us to cover a wide range of values for each hyper-parameter. The best results were obtained with the following regressor parametrization: *'gbtree' booster*, *5-20 max tree depth*, *100 gradient boosted tree estimators*, *regression with squared loss objective*, *0.05-0.1 learning rate*, *'gain' feature importance type*. The initial settings were further adjusted for each of the five algorithms. We used cross validation using five folds on the training set to get a robust performance metric for models evaluation. For the best performing regressor setting for each problem solver, we fitted the model on the entire training dataset.

2) Algorithms runtime prediction: Given the instances' features and the algorithms runtime needed to solve them, we obtained the runtime prediction models. The values of RMSE on cross validation on the training dataset are displayed in Table III.

The large error value of the greedy algorithms, in comparison to the other algorithms results, can be explained by the large runtimes of the greedy approach on the dataset instances. The small error value of the area heuristic, in comparison to the previous algorithms from the table, can also be an effect

TABLE III
THE RMSE VALUES FOR RUNTIME PREDICTION

algorithm	RMSE
genetic algorithm	3.65
greedy	2200
binary heuristic	3.33
area heuristic	0.019
priority heuristic	0.005

of the short runtimes needed by this approach to solve the problems. The priority heuristic approach is also a fast solver for our problem, yielding solutions in a short time span. The runtime prediction model of this heuristic achieved the lowest RMSE on cross validation.

Each model obtained was afterwards run on the test data. For the test dataset predictions, we recorded the actual vs. predicted runtime values in Figure 5. The instances from the test dataset are on the x-axis, while the y-axis contains the execution time for each instance. Take into consideration that the runtime scales of the five plots are different. We also noted the RMSE value for a better comparison of the models' performances. The predictions for the priority heuristic are the closest to their actual values, a fact illustrated also by the small RMSE value. Among all, the largest test error was registered by the greedy runtime model, which could also be anticipated from the previous cross validation results.

E. Solution quality prediction

For the same algorithms, we also trained predictors for the solution quality. The minimal height of the strip required to pack the instances is defined as the solution quality of our problem. Given the characteristics of an instance, the predictors will output an approximation of the result the algorithm is capable of building. For the evaluation of the models' performance, we considered the root mean square error as well.

For the solution quality prediction task, we used an ensemble of models. In this way, we combined different machine learning regressors and returned the average predicted result. The regression models used are: ARDRegression, SVR, Random Forest, and Extra Trees.

The RMSE of the prediction models on cross validation is given in Table IV.

TABLE IV
THE RMSE VALUES FOR SOLUTION QUALITY PREDICTION

algorithm	RMSE
genetic algorithm	170.20
greedy	278.53
binary heuristic	156.96
area heuristic	143.70
priority heuristic	155.80

Figure 6 shows the differences between the predicted and actual result values on the test set.

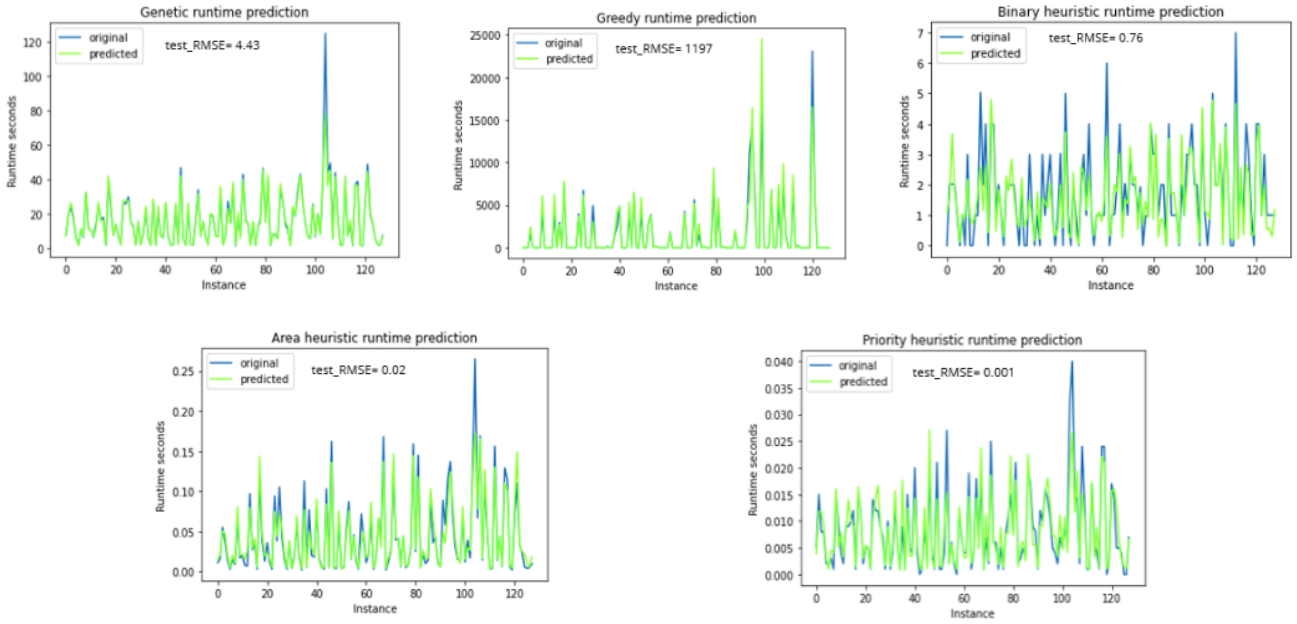


Fig. 5. The performance of the models for the runtime prediction

The evaluation of the prediction model for the greedy result shows the highest RMSE error on cross validation. This can also be observed in Figure 6 as the predictions on the test dataset do not follow so closely the peaks of actual values. With a lower error, the prediction model for the area heuristic result reached the smallest RMSE value on cross validation. This can also be observed by closer predictions to the actual values in the test dataset.

F. Feature importance

An analysis of the input features' importance for the regression models is mandatory in order to establish the feature space in further analysis. We considered the importance of a feature as the average gain across all splits that use the feature. In this way, the importance will reflect the feature's power to group similar instances into child nodes after the split.

For the runtime predictors, the most important features across all predictors are: the number of square items, the total items area, the ratio smallest-biggest and the strip width, as can be observed in Figure 7 (a). The feature importance for the solution quality predictors from the ensemble is presented in Figure 7 (b). The most frequent features across the five models are the baseline height, the total item area, the mean height, and the max item area. Given that the result is the strip height, the features reflect the importance of the height related characteristics for the predictors of the result.

From these insights, it can be observed that there are some features that do not appear to have a significant influence on any model. This leads to the conclusion that not all 23 features are relevant and that a feature reduction technique can be applied without significantly increasing the error value of the predictors. We analysed the effects of applying Principal Component Analysis (PCA) for dimensionality reduction. We ran a model on the original 23 features and obtained a validation error of 0.76. Afterwards, we ran the same model configuration on the first six components after the PCA reduction. We obtained an error of 1.41, respectively 1.25 if we first standardized the features so that their domains did not influence the variance in PCA. In conclusion, a large proportion of the variance is explained by the first six PCA components, but given the light computation of all 23 features, there is no reason to reduce them for this study.

G. Algorithm selection

The algorithm selection system for the two-dimensional strip packing problem is composed of predictors, which estimate the runtime and the performance of each algorithm from the portfolio based on the instance characteristics. Given a new input instance, its features are computed and the regression models predict the output of each solver. The best performing or the fastest approach is selected as the most suitable algorithm for the instance and searched metric.

For evaluating our algorithm selection system, we used the test dataset. On the one hand, for each instance, we computed the best algorithm by taking the minimum value of the actual runs on the algorithms on these instances. This was done to find the actual best algorithms for both the runtime and solution quality metrics. On the other hand, we ran our algorithm selection system on the extracted features of the instances from the test dataset and obtained the predicted best algorithms, one for each metric.

The distribution of the selected algorithms is presented in Figure 8 and Figure 9 as the number of instances from the test dataset for which each algorithm has been chosen

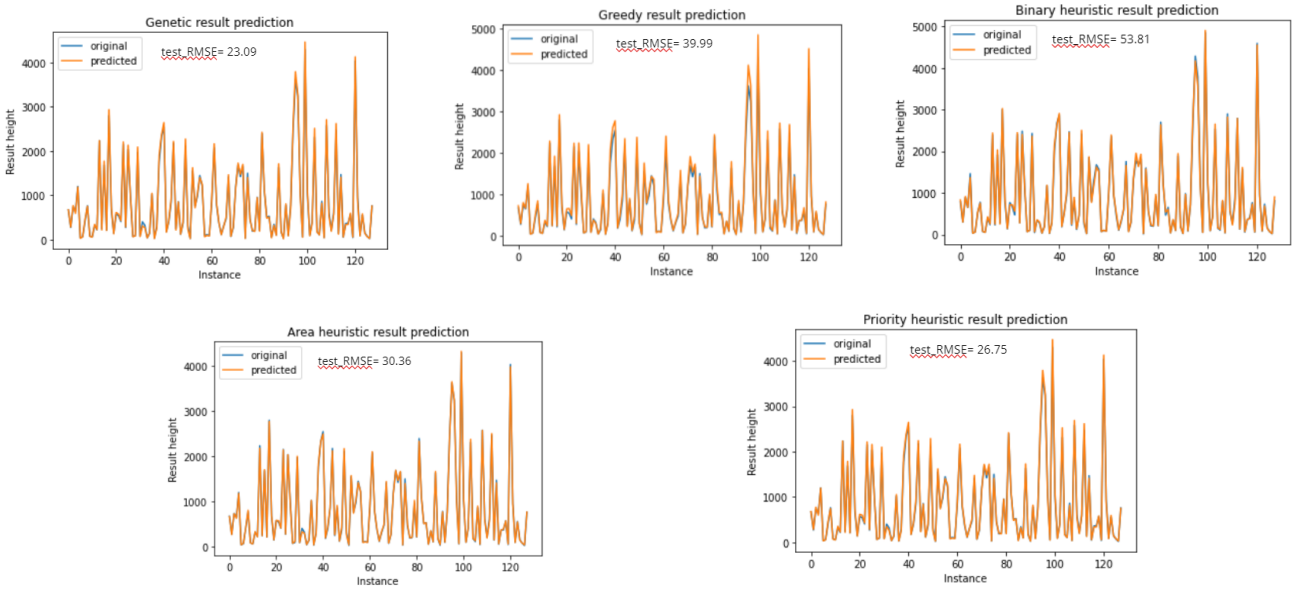
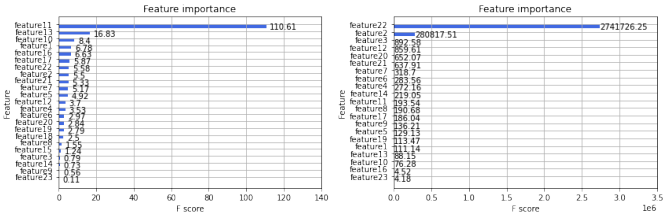
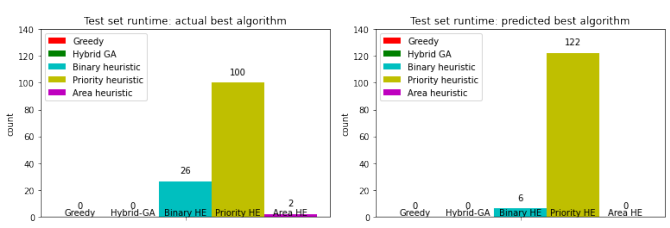


Fig. 6. The performance of the models for the solution quality prediction



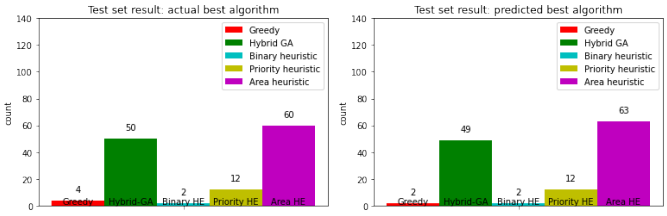
(a) Runtime prediction (b) Solution quality prediction

Fig. 7. Binary approach feature importance



(a) Actual distribution (b) Predicted distribution

Fig. 8. Selected algorithms distribution over the test dataset for runtime criterion



(a) Actual distribution (b) Predicted distribution

Fig. 9. Selected algorithms distribution over the test dataset for result quality criterion

as the best. In Figure 8, the first plot depicts the actual runtime best algorithm distribution, while the second one contains the predicted runtime algorithm distribution. It can be observed that in both cases, only two algorithms were selected, with a slight difference in the proportions. Figure 9 present the selection based on the algorithm result value. For this task, all algorithms are selected for some instances, and the distributions between actual and predicted values are similar.

For a deeper analysis of the results, we performed a per instance comparison between the algorithm label of the best algorithm and the algorithm label selected as the best by the proposed algorithm selection system. The label matching for the runtime algorithm selection task has been recorded as presented in Table V. With a total of 103 correct selections from a total of 128 test instances, our runtime algorithm selection system recorded an accuracy of 80.46%.

Instance	Actual best runtime	Actual best algorithm	Predicted best runtime	Predicted best algorithm	Match
C_10_461	0.007	priority	0.0066	priority	1
C_3_115	0.0	binary	0.004	priority	0
C_10_465	0.005	priority	0.0047	priority	1
...	...	...	...	...	...
C_4_152	0.05	binary	0.1315	binary	1
				SUM	103

TABLE V
ALGORITHM SELECTION BASED ON RUNTIME

For the task of selecting the algorithm with the best result,

the labels matched decreased to 95 out of 128 instances, as shown in Table VI. This leads to an accuracy of 74.21% on the test instances. The decrease can be explained by the distribution of the best performing algorithm on result selection which covers all five algorithms for different instance types, making the task more difficult.

Instance	Actual best result	Actual best algorithm	Predicted best result	Predicted best algorithm	Match
C_10_466	524	genetic	515.1922	genetic	1
C_10_467	644	priority	663.3494	area	0
C_10_491	1415	area	1440.9812	area	1
...	...	...	...	...	...
C_4_152	1432	greedy	1390.1396	genetic	0
				SUM	95

TABLE VI
ALGORITHM SELECTION BASED ON RESULT

Another aspect worth investigating is the optimization of the execution time obtained by running the algorithm selection system instead of the actual five solvers and picking the best result. For the entire test dataset of 128 instances, the algorithm selection system took 6.27 minutes to write the comparison and report the best algorithms for runtime and result metrics. On the other hand, it takes 58.08 hours (3485 minutes) to run the actual five solvers on all 128 instances.

In conclusion, with an accuracy of 75-80% the proposed algorithm selection system is a much faster approach to find the best suited algorithm on an instance basis.

VI. CONCLUSIONS

The paper covers the wide topic of algorithm selection in the context of solving the two-dimensional strip rectangle packing problem. Since there is no algorithm that has the best performance over all instances, we proposed an algorithm selection system. We defined a set of features that could characterize an instance and computed them for our dataset. The features were used to guide regression models towards predicting the instance solution computed by a certain algorithm and the execution time needed to reach it. The proposed algorithm selection system for both execution time and solution quality was built based on the predictors.

To improve the current system, more benchmark instances need to be incorporated. Data can be generated by existing problem generators [23]. The increase of data amount in the predictors' training phase would decrease the error values, while the additional test instances would allow a better evaluation of performance. This might also increase the robustness of the system by adding additional instance types. Finally, a natural direction in the course of the system would be the addition of more solvers for the problem, offering a wider range of packers for the selection.

ACKNOWLEDGEMENT

This paper is partially supported by the Competitiveness Operational Programme Romania under project number SMIS 124759 - RaaS-IS (Research as a Service Iasi).

REFERENCES

[1] J. Augustine, S. Banerjee, S. Irani, Strip packing with precedence constraints and strip packing with release times. In Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures, pp. 180-189, 2006
[2] B. S. Baker, E. G. Coffman, Jr, R. L. Rivest, Orthogonal packings in two dimensions, SIAM Journal on computing, 9(4), 846-855, 1980
[3] J. Jylänki, A thousand ways to pack the bin - a practical approach to two-dimensional rectangle bin packing, retrieved from http://clb.demon.fi/files/RectangleBinPack.pdf, 2010
[4] B. Chazelle, Efficient Implementation, IEEE Transactions on Computers, 32(8), 1983
[5] D. Zhang, Y. Kang, A. Deng, A new heuristic recursive algorithm for the strip rectangular packing problem, Computers & Operations Research, 33(8), 2209-2217, 2006
[6] E. K. Burke, G. Kendall, G. Whitwell, A new placement heuristic for the orthogonal stock-cutting problem. Operations Research, 52(4), 655-671, 2004
[7] A. Gomez, D. de la Fuente, Solving the packing and strip-packing problems with genetic algorithms. In International Work-Conference on Artificial Neural Networks, pp. 709-718, Springer, 1999
[8] S. Illich, L. While, L. Barone. Multi-objective strip packing using an evolutionary algorithm. In 2007 IEEE Congress on Evolutionary Computation, pp. 4207-4214, IEEE, 2007
[9] A. Lodi, S. Martello, & M. Monaci. Two-dimensional packing problems: A survey. European journal of operational research, 141(2), 241-252, 2002
[10] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE transactions on evolutionary computation, 1(1), 67-82, 1997
[11] D. Zhang, L. Shi, S.C. Leung, T. Wu, A priority heuristic for the guillotine rectangular packing problem. Information Processing Letters, 116(1), 15-21, 2016
[12] S. Jakobs. On genetic algorithms for the packing of polygons. European journal of operational research, 88(1), 165-181, 1996
[13] L. Davis, Handbook of Genetic Algorithms, New York, Van Nostrand Reinhold, 1991
[14] J.R. Rice, The algorithm selection problem. In Advances in computers, vol. 15, pp. 65-118, Elsevier, 1976
[15] B.P. Ivanschitz, Algorithm selection and runtime prediction for the two dimensional bin packing problem: analysis and characterization of instances, PhD thesis, Wien, 2017
[16] R.G. Rakotonirainy, A machine learning approach for automated strip packing algorithm selection. ORiON, 36(2), 2020
[17] L. Xu, F. Hutter, H.H. Hoos, K. Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. Journal of Artificial Intelligence Resesearch, 32:565–606, 2008
[18] P. Kerschke, L. Kotthoff, J. Bossek, H.H. Hoos, H. Trautmann, Leveraging TSP solver complementarity through machine learning. Evolutionary computation, 26(4), 597-620, 2018
[19] P. Kerschke, H.H. Hoos, F. Neumann, H. Trautmann, Automated algorithm selection: Survey and perspectives. Evolutionary computation, 27(1), 3-45, 2019
[20] B.E. Bengtsson, Packing rectangular pieces—A heuristic approach. The computer journal, 25(3), 353-357, 1982
[21] J.O. Berkey, P.Y. Wang, Two-dimensional finite bin-packing algorithms. Journal of the operational research society, 38(5), 423-429, 1987
[22] S. Martello, D. Vigo, Exact solution of the two-dimensional finite bin packing problem. Management science, 44(3), 388-399, 1998
[23] E. Silva, J.F. Oliveira, G. Wäscher, 2DCPackGen: A problem generator for two-dimensional rectangular cutting and packing problems. European Journal of Operational Research, 237(3), 846-856, 2014
[24] D. Johnson, Near-Optimal Bin Packing Algorithms. Ph.D. Thesis, Massachusetts Institute of Technology, Dept. of Mathematics, 1973 (2010)
[25] J.A. George, D.F. Robinson, A heuristic for packing boxes into a container. Computers & Operations Research, 7(3), 147-156, 1980