Noname manuscript No.
(will be inserted by the editor)

Application of supervised machine learning methods on the multidimensional knapsack problem

Abdellah Rezoug · Mohamed Bader-el-den · Dalila Boughaci

Received: date y / Accepted: date

Abstract Machine Learning (ML) has gained much importance in recent years as many of its effective applications are involved in different fields, healthcare, banking, trading, gaming, etc. Similarly, Combinatorial Optimisation (CO) keeps challenging researchers by new problems with more complex constraints. Merging both fields opens new horizons for development in many areas. This study investigates how effective is to solve CO problems by ML methods. The work considers the Multidimensional Knapsack Problem (MKP) as a study case, which is an np-hard CO problem well-known for its multiple applications. The proposed approach suggests to use solutions of small-size MKP to build models with different ML methods; then, to apply the obtained models on large-size MKP to predict their solutions. The features consist of scores calculated based on information about items while the labels consist of decision variables of optimal solutions calculated from applying CPLEX Solver on small-size MKP. Supervised ML methods build models that help to predict structures of large-size MKP solutions and build them accordingly. A comparison of five ML methods is conducted on standard data set. The experiments showed that the tested methods were able to reach encouraging results. In addition, the study proposes a Genetic Algorithm (GA) that exploits ML outputs essentially in initialisation operator and to repair unfeasible solutions. The algorithm denoted GaPR explores the ML solution neighbourhood as a way of intensification to approach optimal solutions. The carried out experiments indicated that the approach was effective and competitive.

Keywords Machine learning · multidimensional knapsack problem · genetic algorithm · combinatorial optimisation · evolutionary computation

A. Rezoug
Department of Computer science
University Mhamed Bougara, Boumerdes, Algeria
E-mail: abdellah.rezoug@gmail.com

M. Bader-el-den
School of computing, University of Portsmouth
Portsmouth, United Kingdom
E-mail: mohamed.bader@port.ac.uk

D. Boughaci
Department of Computer science
University of Science and Technology Houari Boumedienne
Algiers, Algeria
E-mail: dalila_info@yahoo.fr

1 Introduction

Machine learning (ML) is the field of computer science that focuses on collecting, filtering, analysing data to find patterns and structures to enable predicting, reasoning, and decision making independently of human interaction. During the last few years, there has been a remarkable interest in this field, and people are more and more used to the ML applications. Image and speech recognition, video surveillance, sentiment analysis, medical services (disease identification and risk satisfaction), product recommendations, virtual personal assistants (Siri, Alexa), email spam and malware filtering, etc. are the most common applications of ML.

ML is also employed to solve some of well-known combinatorial optimisation problems. General surveys about ML using for CO problems are given by [21, 6, 20]. Also, an interesting survey on learning-metaheuristics is presented by [29]. The mathematical representation of Graph-nature CO problems facilitate to practice simple ML manipulation backed by simple or

sophisticated algorithms. The routing and networking problems such as the Travelling Salesman Problem (TSP), the Vehicle Routing Problem (VRP), etc. have simple features easy to represent and their solution are of the same nature. However, data representation is a real issue when trying to deal with many CO problems by ML theories. Few works are done for other kind of CO problems because representing data, solution and objective function is not compatible with ML theories.

Another problem concerns generalising models to more or larger data. In fact, when a model is built on data, it is likely hard, sometimes impossible to generalise it for other than the same kind of data or even on larger data. However, some recent researches succeeded to develop techniques to tackle this issue but for one or two CO problems.

Nowadays, researchers in different scientific fields are interested in machine learning applications, however, the use of ML concepts in the case of CO problem has been poorly investigated. This paper intends to investigate whether ML methods are applicable or not to deal with CO problems. This investigation takes the form of a case-study where the Multidimensional Knapsack Problem (MKP) is studied. Five ML algorithms will be examined, all of which are a regression machine learning. Consequently, two main contributions are proposed: The first aims to create ML models from small-size MKP instances. The models are then applied following a five-step procedure to build solutions to large-size MKP. A Cross-validation and hyper-parameters tuning processes are conducted on training and test data to avoid underfitting and overfitting problems. The second integrates predictions of the best model in a Genetic Algorithm (GA) variant. Two of the genetic operators are enhanced by ML finding 1) The initialisation operator is mainly based on the solutions predicted by the ML procedure 2) The repair operator that fixes unfeasible solutions also uses the predicted solution to complete the resulting feasible solution.

The approach uses results obtained from solving small-size MKP instances by CPLEX to build models. As feature, a mathematical equation is used to calculate the score for each item which is based on the value (or price) and the weight of the item. Also, the approach considers the decision variables given by optimal solutions as the labels. We use data created in the same way composed of simple and complex instances. Simple data are used to build models which we use to predict if the item is likely to be selected regarding its score and then build the solution by a naive method.

This paper is composed of two parts: firstly, we analyse and compare five ML methods effectiveness on MKP. The following supervised regressive methods are involved: Bayesian ARD regression (BAR), Gaussian process regression (GPR), k-nearest neighbours regression (NNR), random forest regression (RFR) and epsilon-support vector regression (SVR). A comparison revealed that the GPR method performance was the best. Secondly, we incorporate the best method (i.e. GPR) as prior analysis to predict solution and then using it to initialise the population of a Genetic Algorithm (GA). The proposed algorithm named GA Combined with GPR (GaPR) is examined across standard data set and the obtained results are compared to the state-of-the-art. The experiments showed encouraging results in terms of the fitness function.

The remaining of this paper is organised as follows: Section 2 provides a review of some ML approaches used to deal with CO problems. Section 3 describes the multidimensional knapsack problem (MKP). The application procedure of some ML on MKP as well as the results are given in Sections 4 and 5 respectively. Section 6 and 7 present the GaPR variant and the obtained results from its application on standard data set. The conclusions are given in Section 8.

2 Related works

Several approaches have been proposed to deal with CO problems, some focusing on exact methods, others on approximate methods, however, few pay particular attention to ML except to tune algorithms parameters. Although many authors have conducted studies, this field is still insufficiently explored. This section attempts to survey the state-of-the-art of the aspect including the different classes of solutions. Three classes are distinguishable on how authors utilised ML methods alongside with CO methods in many areas 1) solving ML problems by CO methods 2) cooperation between ML and CO algorithms for CO problems 3) ML methods directly dealing with CO. This section focuses on the third class of approaches.

ML has limited capacity to directly solve discrete optimisation problems and needs further steps to formulate the knowledge they give to build feasible solution. However, it has been proven in many cases that this way of proceeding is very effective. The idea behind is to train ML model to predict solutions directly from the input solutions.

The application of ML to CO problems is not new. Neural networks have been widely studied in the past and even nowadays on standard CO problems such as TSP, VRP, Knapsack Problem (KP), etc. Vinyals et al. [30] proposed a supervised Recurrent Neural Network (RNN)-based method as a solution to some CO

problems with special focus on TSP. A sequence 2 sequence model called pointer network uses RNN encoder and decoder (Long Short Term Memory (LSTM) and attention Mechanism [2] respectively) to produce an encoding for each node of a TSP graph to predict future movements. The advantage of this method is that it can be generalised to different sizes of graphs. Bello et al. [5] suggested a platform based on Reinforcement Learning (RL) and the pointer network model. The pointer network model is driven by a RL based on the Actor-Critical algorithm [23] and negative turn length as reward. The RL is expensive in computational time but allowed to obtain solutions very close to optimal and has surpassed supervised methods.

Although the RNN architecture is able to consider some characteristics of graph-nature problems, it is not able to take into account all the characteristics especially the relationships between the different entities of a graph. The Graph Neural Network (GNN) is the tool that offers this advantage and allows to represent more information about the problem. Therefore, many researchers have recently focused on this approach. Kool et al. [17, 16, 18] utilised the Transformer's architecture [24] to propose a GNN-based focus model to address several routing problems: TSP, VRP and Orienteering Problem (OP) by introducing many prior knowledge to the model. An attention mechanism is used to represent the relationship between the graph nodes. The results are interesting for small data only. A Graph Convolutional Network (GCN) model [15] was applied with a tree-guided search algorithm to address the problems of maximum independent set and minimum summit coverage [19]. Nazari et al. [25] proposed a modified pointer network architecture to address the VRP problem. Khalil et al. [14] proposed an end-to-end machine learning framework that combines deep graph embedding network (called structure2vec (S2V) [13]) with reinforcement learning (q-learning). By learning greedy algorithms, the framework automatically designs heuristics for three graph-based combinatorial optimisation problems (Travelling Salesman Problem, Minimum Vertex Cover and Maximum Cut). The experiments showed that the framework performed better than manually designed greedy algorithms across different problems and graph types and sizes. As an extension of the work by [5], Emami et al. [10] have proposed deep reinforcement learning (DRL)-based methods for solving combinatorial problems. A data-driven algorithm for learning permutations based on a Sinkhorn Policy Gradient (SPG) produces continuous relaxations of permutations matrices to train an actor-critic architecture. For further details on ML usage for CO problems, the reader may refer to [21, 6, 20] where detailed surveys are provided.

3 Multidimensional Knapsack Problem (MKP)

The MKP is an important np-hard CO problem largely studied to test optimisation approaches in operational research and computer sciences. Moreover, it models many real-life applications in a different field such as resource allocation in distributed environment [12], resource allocation of adaptive multimedia systems [1], capital budgeting [22], project selection [4], etc. The MKP is composed of n items and one knapsack with m different capacities c_i where $i \in \{1, \ldots, m\}$. Each item j where $j \in \{1, \ldots, n\}$ has a profit p_j and can take w_{ij} of the capacity i of the knapsack. The goal is to pack the items in a way that maximises their profits without exceeding the knapsack capacities. the MKP can be represented as the following integer program:

$$MKP = \begin{cases} max & \sum_{j=1}^{n} p_j x_j \\ sub\ to & \sum_{j=1}^{n} w_{ij} x_j \leq c_i \quad i \in \{1 \ldots m\} \\ & x_j \in \{0,1\} \qquad\quad j \in \{1 \ldots n\} \end{cases}$$

In practice w_{ij} is represented with an integer matrix n x m where a line j corresponds to the weights of item j on each of the n dimensions. Whereas a column i is the weights of every item on dimension i. In addition, p_j is represented with an n integer vector where every single number represents the value of the corresponding item (value could be a priority or price). Solving MKP consists to find the subset of items that maximises the overall p_j without exceeding the knapsack capacities. Besides, c_i is represented with an m integer vector which corresponds to the knapsack dimensions capacities.

A feasible solution X for the MKP represents the selected items to be packed in the knapsack. The decision variables x_j are binary where $x_j = 1$ means that the item j is packed in the knapsack, and $x_j = 0$ means that item j is not packed in the knapsack. w_{ij} represents the weight of the item j on the dimension i.

4 Procedure of MKP solving with ML methods

This section describes the procedure to build a solution from the small-size MKP instances to its solution. The procedure is divided into five steps. The small-size MKP instances and the CPLEX Solver are used to create labels and features for the ML methods to build the models. Then, the produced models are used for complex instances to predict information about their solutions. The solutions are then built using a naive method.

Step 1 label

Small-size MKP instances (i.e. n = 100) are solved to the optimum using CPLEX. The CPLEX Solver is a mathematical optimisation software developed by IBM. It implements many well-known methods such as: Simplex, branch and bound, branch and cut, etc. capable to deal with integer programming problems and very large linear programming problems. Besides, CPLEX provides interfaces for interacting in different programming languages (Python, C++, Java, etc.). The Java interface is the one used in this study to solve small-size MKP instances. So, a model is defined composed of two elements: the objective function in which the decision variables are defined as binary and the list of constraints. Then, the solver is invoked to start the optimisation process. Finally, the solutions represented by the decision variables are recorded such that each item corresponds to its decision variable. Each solution is represented with a binary vector where 1 means a selected item and 0 a left item. This binary vector will be considered as a label for the input of ML methods in the rest of this work.

Step 2 feature

One feature is considered as MKP has a relatively simple mathematical model. An item is defined by a weight vector relative to the knapsack dimensions and a value or a price (see Section 3). These two criteria act differently on the choice of items, an item that maximises the objective function has a better probability of being selected if it has high price (value) and low overall weight. In order to distinguish the best items from the worst, the two criteria are translated by the price / weight $score_j$ (score of item j) as shown by equations 1-2 bellow :

$$score_j = \frac{p_j/mean(p_j)}{e_j/mean(e_j)} \qquad j \in \{1 \dots n\} \qquad (1)$$

$$e_j = \sum_{j=1}^{m} w_{ij}(\sum_{l=1}^{n} w_{il} - c_i) \qquad i \in \{1 \dots m\} \qquad (2)$$

To take into account the contribution of the constraints (weights) Senjo and Toyoda [28] calculated e_j (equation 2). This formula allows to avoid that a subset of constraints dominates all others [26]. Furthermore, to calculate the *score*, one of e_j and p_j could dominate the other, dividing by the mean numbers makes all measurements meaningful and close (i.e. values in [0, 2]). Thus, makes the *score* data a correct ML feature.

In this study, a $score_j$ vector is calculated for each MKP instance and is considered as the only feature. At this stage, each item has a feature and is labelled following the first step. Therefore, supervised ML methods are applicable on this input and are able to construct models. Figure 1 shows the distribution of items score of six MKP considered to make the feature in this study (more detail is given in section 5.1).

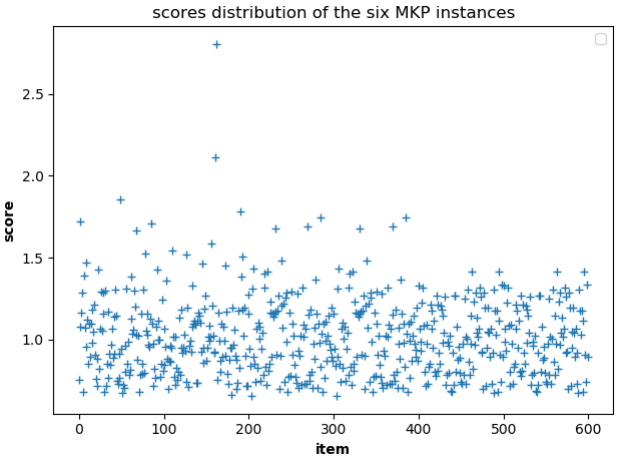


Fig. 1: score values distribution

Step 3 model

In the context of combinatorial optimisation, it is probably less important to follow the known approach of machines learning, because the purpose of solving problems is to optimise an objective function. Therefore, the accuracy of an ML method is measured by how close the achieved solution is to the optimum. Five regression methods of ML are examined in this study. The input of these methods is formed by data composed of features and their labels. A ML method produces a model that will be used to predict how likely an item is to be selected. The five ML methods involved in this study are: Bayesian ARD regression (BAR), Gaussian process regression (GPR), k-nearest neighbours regression (NNR), random forest regression (RFR) and epsilon-support vector regression (SVR).

Step 4 prediction

After calculating the models, they are used to predict outputs for large-size MKP instances. For each item a real value is given which represents the model's prediction of its score. In different context, applying classification methods returns one of two classes, either item selected or not selected (i.e. binary classification). But, regression methods predict a continuous value for each

input that may help to take decision about whether packing or not an item. Thus, for each MKP instance, a vector containing the scores of items is entered for each model, and another vector containing the resulting predictions is produced. Then, these predictions are used to calculate a feasible solution to the corresponding MKP problem.

Step 5 solution

An MKP solution is built by a naive method as following:

1. Sort items in descending order regarding their predictions.
2. Create S an empty solution.
3. Add the items one by one to the solution S, as long as all constraints are satisfied to keep the solution feasible.
4. Calculate the 'distance from the optimum' or GAP criteria of the solution S.
5. Calculate the fitness function value.
6. The result S is a feasible solution.

5 MKP solved with the five methods

Empirical performance evaluations are of great benefit, as they emphasise the strengths and weaknesses of the evaluated methods and thereby guide research efforts into promising directions. This section presents a comparison between the ML methods, and this is done with respect to the GAP, a method is better if its GAP is low. The standard OR-Library data often used by researchers in the field are used to do this study. It is composed of 270 instances of large-size MKP whose number of constraints m is in $\{5, 10, 30\}$ and the number of variables n is in $\{100, 250, 500\}$. But to validate the models we need first to conduct a hyper-parameters tuning and a cross-validation processes.

5.1 Hyper-parameters fitting and cross-validation

When using ML methods, it is difficult to know exactly what are the best hyper-parameters of a given algorithm on a given problem, therefore it is common to apply random or grid search methods to fit them. Thus, the grid search strategy is applied to perform the hyper-parameters optimisation. For that, six small-size MKP instances have been chosen to create the data with: n = 100, m = {5, 10, 30} alpha = 0.25 and instances {1, 2}, so the instances are : {5x100-0.25-1, 5x100-0.25-2, 10x100-0.25-1, 10x100-0.25-2, 30x100-0.25-1, 30x100-0.25-2}. Additionally, the data is divided into training set 70% and test set 30%. A K-fold cross-validation method is also considered on training set with k = 4. The score of the obtained models on test set was recorded and Table 1 presents the best parameters and gives the score R2 on test sets of each method. Furthermore, the *mean squared error regression loss "mserr"* and the *maximum residual error "merr"* have been measured.

According to the score R2, NNR model was able to represent 83% of the test set, then GPR 70%, RFR 69%, SVR 61% and finally BAR 53%.

5.2 Comparison between the five models

Regression analysis was used to predict the outputs of the respective models to the items of each MKP problem. Table 2 compares the results obtained from the preliminary analysis of the five methods. The average GAP of each of the nine groups of data-set is examined. The average GAP of each method, calculated using the procedure explained in Section 4, were compared in order to define which one achieves the best results. In order to show the differences in terms of performing, detailed comparison is given in Table 2 and Figure 3.

It can be seen from the data in the histograms of Figure 2 that, considering the performing, the five methods can be classified into three subsets. Firstly GPR and BAR, these two methods allowed to calculate the best MKP solutions for all the nine groups. Secondly, SVR was capable to predict high-quality solutions also and its results are not as good as the first two algorithms. Finally, RFR and NNR results are of inferior quality than the others on all the nine groups of data.

Closer inspection of the data shows that all the methods allowed to build high-quality solutions with low GAP. More specifically, GPR gave the lowest GAP, it is the best performing method with an overall GAP less than 5% for almost all the nine groups except the groups 4 and 7, for which the overall GAP was less than 5.5%. The results indicate that BAR obtained the second-best performance and reached better solutions than the remaining methods with an overall GAP less than 4% for groups 2, 3, 5, 6 and 9 and less than 6% for the others. Besides, SVR was able to reach good results. RFR and NNR had a comparative performing with an overall GAP between 3% and 7%. They achieved the worst results on all the data-set compared to the others.

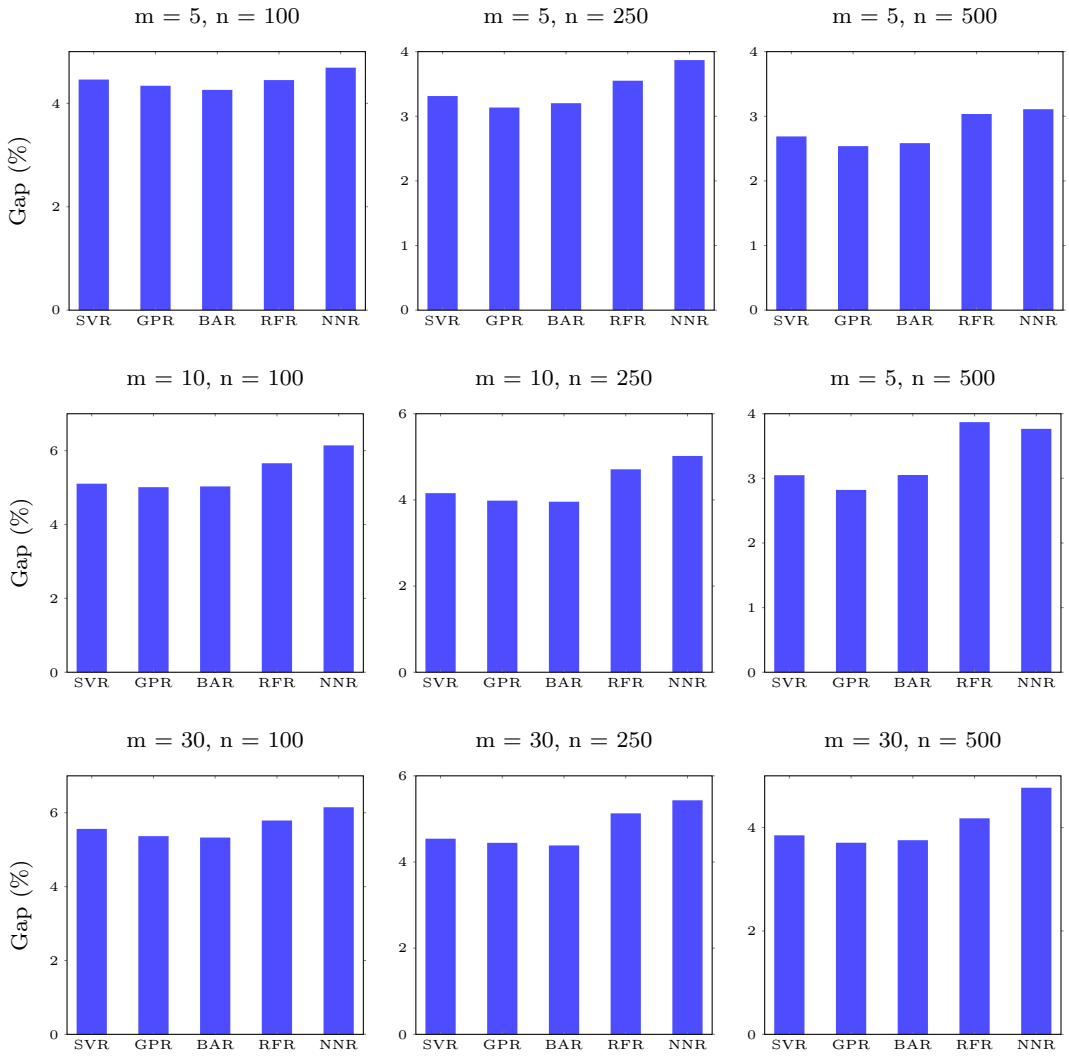


Fig. 2: Comparison of the five methods performances for some MKP instances.

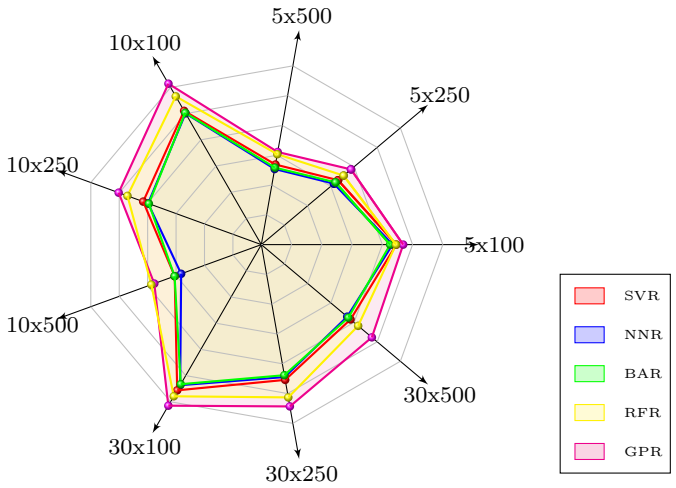


Fig. 3: Comparison of the five machine learning methods in terms of average GAP on the 9 groups of mkp dataset

method	Best parameters	$R^2(\%)$	$merr$	$mserr$
SVR	$C = 1.0, cache\_size = 200, coef0 = 0.0, degree = 3, epsilon = 0.001, gamma = auto\_deprecated, kernel = rbf, max\_iter = -1, shrinking = True, tol = 0.001$	61	0.92	0.08
BAR	$alpha\_1 = 1e-06, alpha\_2 = 1e-05, lambda\_1 = 1e-05, lambda\_2 = 1e-05, n\_iter = 200, tol = 0.0001$	53	1.62	0.09
NNR	$algorithm = brute, leaf\_size = 1, n\_neighbors = 15, p = 2, weights = distance$	83	0.86	0.03
RFR	$n\_estimators = 500, min\_samples_split = 20, min\_samples\_leaf = 2, max\_features = auto, max\_depth = 110, bootstrap = True$	69	0.92	0.06
GPR	$alpha = 0.1, kernel = RBF(length\_scale = 10)$	70	0.86	0.06

Table 1: Hyper-parameters fitting, cross-validation Score $R^2(\%)$, maximum residual error $merr$ and mean squared error regression loss $mserr$ of the five ML.

m	n	SVR	GPR	BAR	RFR	NNR
5	100	4,46	4,34	**4,26**	4,45	4,69
	250	3,31	**3,14**	3,20	3,55	3,87
	500	2,69	**2,54**	2,58	3,04	3,11
Group average		3,49	**3,34**	3,35	3,68	3,89
10	100	5,11	**5,01**	5,03	5,66	6,14
	250	4,16	3,98	**3,96**	4,71	5,02
	500	3,05	**2,82**	3,05	3,87	3,77
Group average		4,10	**3,94**	4,01	4,75	4,98
30	100	5,56	5,37	**5,33**	5,79	6,15
	250	4,54	4,45	**4,38**	5,13	5,43
	500	3,85	**3,71**	3,76	4,18	4,77
Group average		4,65	4,51	**4,49**	5,03	5,45
	Overall average	4,08	**3,93**	3,95	4,49	4,77

Table 2: The average GAP from the five ML methods application on the data set.

What stands out in the experiment is that: firstly, ML methods can successfully solve MKP with encouraging results. Secondly, only one feature about MKP simple data set were useful to build models capable to predict solutions of complex data of the same problem. Thirdly, the execution time to calculate solutions of large-size MKP instances by using ML methods is insignificant than when other methods are used. Finally, searching in the neighbourhood of the obtained solutions is likely to approach the optimal solutions.

5.3 Behaviour of the five methods on data

Figure 4 represents nine cases of MKP chosen semi-randomly, for each case, the distribution of 'predicted regression' is plotted as a function of *score* and this for all the five ML methods implied in this work. This experiment aims to inspect the way each model of the ML methods represents the prediction and how the models are built out of the data. The scikit-learn [1] library of Python language has been utilised to draw the distributions.

From the distributions of predictions, it can be noticed that SVR and GPR obtain almost similar representation even when complexity increases. In fact, by calculating the feasible solution of the respective MKP, both methods were able to construct comparable predictions. Prediction values for both methods vary in [0, 1] with more density around 0. Also, BAR predictions take a linear form as almost all predictions are on the same line.

In addition, NNR has a model that act like a classification method. The prediction gives three classes of results: a first class of items with a prediction equal to 0, a second class of items with prediction equal to 1 and a third class between 0 and 1. Moreover, NNR classifies items into two main classes for case (a), class 1 and class 0. Aside this case, NNR gets outputs close to 0 for most of the scores of the instances included in this study. Also, the distribution of predictions when using

[1] https://scikit-learn.org/stable/

RFR does not take any graphical form, in fact almost all points are placed randomly between 0 and 1.

6 Genetic Algorithm combined with GPR (GaPR)

ML method itself does not find optimal solutions, and there is no way to introduce MKP constraints into its procedure. In addition to that, it is a statistics-based mechanism whose purpose is to support decision making. However, the information it offers about items can be very useful for supporting algorithms dedicated to this operation, consequently, the heuristics are the most suitable here.

Genetic Algorithm (GA) is a heuristic inspired by evolution and natural selection of living species. It is based on a population of individuals that evolves in iterative way by operations (or operators) of crossing and mutation between the elites of individuals, at the same time bad individuals eventually disappear during the passage to the following generations. GA has a lot of applications and has many variations since its inception aimed to improve its performance and making it faster.

GA has been widely proposed in the form of MKP solution approaches, whether alone or hybridised with other heuristics or exact methods. In this part, the GA Combined with GPR (GaPR) is proposed. GaPR incorporates GPR solutions in GA firstly, to create the initial population and secondly, to repair unfeasible solutions. In GaPR solution, GA is combined with GPR method which represents a novelty and opens the way to further research in this perspective.

In view of the results obtained previously, the comparison showed that GPR method achieved the best solutions. Knowing that these are close to optimal solutions, they are integrated into the creation of the initial population of a GA. The idea is that since the optimal solutions are not far away it is enough to search around these solutions to find promising solution areas.

6.1 Initial population

The individuals of initial population are created with a special procedure that implies the GPR solution and $X1\_core$ items as follows: A probability r is generated randomly and if it is less than a pre-defined GPR rate gr, then the procedure randomly chooses an item from the GPR solution (We make sure that this probability is high). Otherwise, the procedure chooses an item from $X1\_core$. $X1\_core$ is a vector of items made of:

1. $X1$ which is a feasible solution built with a simple method from the items sorted decreasingly regarding their $efficiency$ as follows:
   (a) The items are sorted based on their $efficiency$ (equation 3) [28]. The items at the top of this list are likely to be selected while the items at the bottom are unlikely to appear in good solutions.
   $$efficiency_j = \frac{p_j}{\sum_{j=1}^{m} w_{ij}(\sum_{l=1}^{n} w_{il} - c_i)} \quad (3)$$
   (b) $X1 \leftarrow \emptyset$
   (c) Add the sorted items one by one to $X1$ while all constraints are satisfied.
   (d) Stop if at list one constraint is broken.
2. $core$ which contains the few items that follow the last item of $X1$. The $core$ size depends on the MKP size.

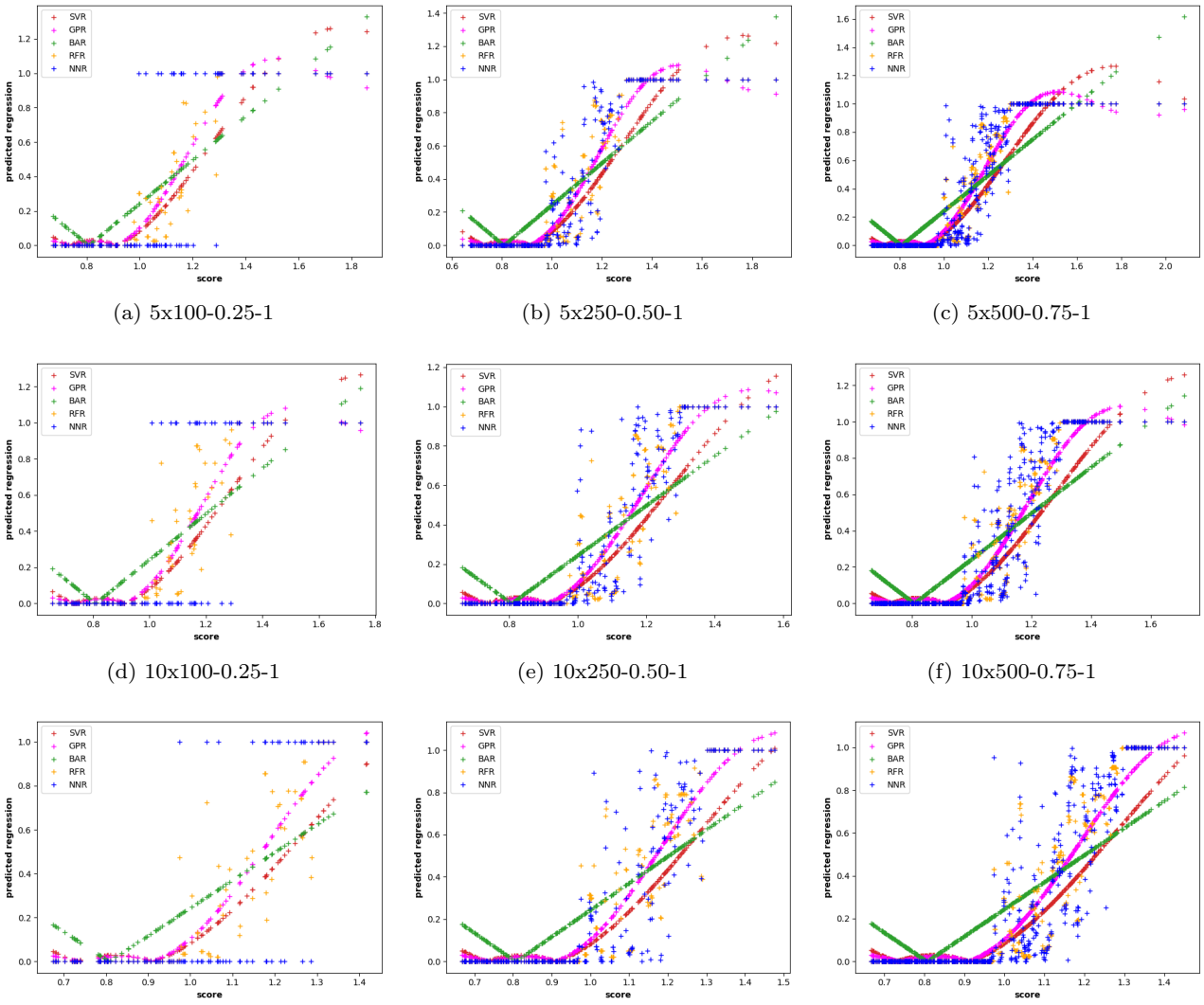
The pseudo-code of Algorithm 1 shows the algorithm used to create the initial population:

Algorithm 1: Initialisation operator

```
input : GPR solution (gs), X1_core, MKP
        instance (mkp), population size (ps),
        GPR rate (gr), number of iteration (ni)
output: an initial population (pop)
pop ← ∅ ;
for i ← 1 to ps do
    s ← ∅;
    // create s a new solution
    for j ← 1 to ni do
        r ← random real in [0, 1];
        // random rate: generate a random
           probability
        if r < gr then
            item ← a random item from gs;
        else
            item ← a random item from X1_core;
        if (s + item) verifies constraints then
            s ← s + {item};
            // if adding item to s keeps it
               feasible
    pop ← pop + s
return pop
```

6.2 Genetic operators

The selection operator is based on *Tournament* algorithm where a subset of individuals of known size is chosen randomly and then the two best individuals (in terms of fitness) are selected. The crossing is performed on the two individuals selected according to the multi-point crossing algorithm. An exchange of multiple items between the two individuals while checking the MKP



(a) 5x100-0.25-1 (b) 5x250-0.50-1 (c) 5x500-0.75-1

(d) 10x100-0.25-1 (e) 10x250-0.50-1 (f) 10x500-0.75-1

(g) 30x100-0.25-1 (h) 30x250-0.50-1 (i) 30x500-0.75-1

Fig. 4: The predicted regression through the score on some MKP instances for the five methods

constraints and the crossover rate determines the number of items exchanged. Two new individuals are built to whom a repair is applied to make them feasible solutions. The repair algorithm is described in the next section. The mutation operator is simple where each new individual undergoes a simple mutation such that a single item is removed from it, and a repair operation is applied to it.

6.3 Repair operator

The repair takes place in two stages, the first makes the solution feasible by removing all eventual conflict and the second tries to complete the solution by adding items.

1. **Solution feasibility:** as long as at least one constraint is not verified the procedure removes several times an item according to a probability either: having the lowest score or randomly, knowing that the first option is much more likely than the second.
2. **Adding items:** to complete a solution, the items, sorted according to their scores and the items of the best individual in the population are utilised. The procedure check them and each item that does not cause conflict is added to the solution. Consequently, items that maximise the objective function are favoured with this procedure.

The pseudo-code of Algorithm 2 details the repair operator.

Algorithm 2: Repair operator

input : MKP solution s, the currant best solution in *pop bsts*, sorted items si, ML delete rate (mlr)
output: s feasible and improved

Phase I: fixing s

```
while (s unfeasible) do
    r ← random rate();
    // random rate: generate a random
       probability
    if r < mlr then
        w_item ← worst_item(s);
        s ← s − {w_item};
        // Remove the item having the worst
           score
    else
        s ← s − {random_item};
        // Remove a random item
```

Phase II: improving s

```
ir ← a random real in [0, 1];
if ir < bsts then
    foreach item it of the vector si do
        if s + {it} keeps s feasible then
            s ← s + {it}
else
    foreach item it of the bsts do
        if s + {it} keeps s feasible then
            s ← s + {it}
return s
```

The GaPR algorithm is composed of two phases: model creation and Genetic operators. It is explained as follows :

Phase 1. model creation

1. Determine *data* : a subset of k MKP with $n = 100$ and $m = 5$ and $\alpha = 0.25$.
2. For each MKP in *data*, calculate the *score* vector according to equation 1. The feature of the GPR is composed of all these score vectors $feature = \{score_1, score_2, ..., score_k\}$.
3. Solve each of data MKP instances using CPLEX solver. From this operation result k binary vectors that represent the solutions *sol*. Together they form the GPR target $target = \{sol_1, sol_2, ..., sol_k\}$.

	$score_1$				$score_2$				...		$score_k$			
feature	0.7	0.7	0.3	0.1	0.1	0.3	0.9	0.2	..	..	0.3	0.5	0.9	0.1

	sol_1				sol_2				...		sol_k			
target	1	1	0	0	0	0	1	0	..	..	0	0	1	0

4. The *model* is then built from applying the GPR on the input composed of $(feature, target)$.

Phase 2. genetic operators

1. For an MKP instance I calculate the score vector $score_I$.
2. Calculate the model prediction of the $score_I$ by applying the *model*.

$score_I$	0.65	0.78	0.03	0.7
$prediction(score_I)$	0.4	0.8	0.1	0.7

3. The items are then sorted according to their prediction $prediction(score_I)$. After that, a feasible solution $Psol$ (for predicted solution) is created by adding the items respecting the order and of course the problem constraints.
4. **Initialisation operator**: it is detailed by the pseudo-code of Algorithm 1.
5. **Selection operator**: it follows the tournament selection algorithm where a tour of individuals are selected randomly and from them two are chosen to be the parents.
6. **Crossover operator**: the multi-point crossover method is used to build two offspring *offspring1*, *offspring2* from the selected parents.
7. **Mutation operator**: with a mutation probability, replace one item from each $offspring$ by another both chosen randomly.
8. **repair (offspring1), repair (offspring2)**. The repair pseudo-code is given by Algorithm 2.
9. replace the two worst individuals in the population by $offspring1$ and $offspring2$ if their fitness are better.
10. Go to 9 if the stopping condition is false.

7 Experimental results of GaPR on MKP

This section presents the results of applying the GaPR on standard data and the average fitness over 30 runs is calculated. The data benchmark employed to undertake experimental tests are the standard OR-Library data set presented by Chu and Beasley [7] composed of 270 instance divided into 9 classes of 30 instance each. Where the number of items n is in {100, 250, 500} and the number of constraints m is in {5, 10, 30 }. For each combination (n, m), the constraint capacities are computed as $\alpha \sum_{j=1}^{n} w_{ij}$, where α is a tightness ratio. For the first 10 instances, $\alpha = 0.25$; for the next 10 instances, $\alpha = 0.5$; and for the remaining ones, $\alpha = 0.75$.

The GaPR parameters were chosen in a way to concentrate the search at the neighbourhood of the GPR solution. Table 3 summaries the parameters values utilised in this work.

The results from applying GaPR on the 270 MKP instances are summarised in Table 4. The average distance from optimum 'GAP' values of each 10 instances

Parameter	Description	n= 100	250	500
ps	population size	60	60	60
ni	number of iteration	200	500	1000
gr	GPR rate	0.9	0.9	0.9
cs	core size	15	25	40
mlr	ML delete rate	0.95	0.95	0.95
cr	crossover rate	0.2	0.1	0.05
mr	mutation rate	1	1	1
tour	tour of selection	5	5	5
bsts	rate best solution	0.15	0.15	0.15

Table 3: The GaPR parameters

is given knowing that each instance is executed 30 times to obtain the average GAP. What is interesting about the data in this table is that GAP is very close to zero which means that the algorithms constantly reach close-to-optimal solutions even when n and m increased. Closer inspection of the table shows that the algorithm performed better on the instances where the correlation α between weight and value is less tight.

m	α	n = 100	n = 250	n = 500
5	0,25	0,09	0,22	0,31
	0,50	0,08	0,17	0,25
	0,75	0,05	0,16	0,76
	Group average	0,07	0,18	0,44
10	0,25	0,32	0,44	0,64
	0,50	0,17	0,32	0,46
	0,75	0,12	0,28	0,89
	Group average	0,2	0,34	0,66
30	0,25	0,63	0,96	1,42
	0,50	0,28	0,54	0,92
	0,75	0,14	0,44	1,26
	Group average	0,35	0,65	1,2
	Overall average	0,21	0,39	0,77

Table 4: Summary of the average GAP (%) from applying the proposed approach on the data set.

In order to analyse the performance of GaPR when α, n and m change, we conducted a statistical experiment. Randomly, we have picked the 5th MKP instance out of each of the 27 classes (no rule is applied to make this choice and all conclusions are valid no matter the instance number). Then, GaPR was applied on each instance thirteen (13) times and each time the gap from optimal solution was recorded. These records permeated to calculate the Best, the Worst, the Mean the first and the third quartile solutions in order to draw boxplots, the results are given by Figure 5. The boxplots are presented in such a way that allow to compare the GaPR performing according to the target parameters. From observing each box alone, we can understand the GaPR behaviour when α changes in $\{0.25, 0.50, 0.75\}$. Whereas, observing the boxes vertically compares how it behaves when the number of constrains m increases in $\{5, 10, 30\}$. Similarly, observing them horizontally, explains how its performing is when the number of items n increases in $\{100, 250, 500\}$. The comparison for the two former cases should be for the same α.

From the results, the first finding is that GaPR keep giving high-quality performance even when the MKP dimensions increase significantly. Concerning α, the results show that GaPR performs better when α increases, however, the gap is always close to zero. Regarding m, the GaPR performing remains excellent even when the number of constraints is 30, and it is better with less constraints. We can see that GaPR obtains less good results when the number of items increase to 500. The worst performance can be seen in the seventh box ($m = 30$, n= 100 and $\alpha = 0.25$).

The results obtained from the application of GaPR on the MKP instances with $n = 500$ are summarised in Table 5 (the first three instances of each class). Many data are recorded from 30 runs on each instance where : $averagef$ the average fitness, $fbest$, $fworst$ the best and the worst recorded fitness, PSG the gap from the optimum of the GPR solution, *gap* the gap from the optimum of the average fitness, and the *gapbest* gap of the best obtained solution and the optimum.

The results indicate that in 16 of 27 cases the GPR solution gap is less than 3%, in 10 cases between 3-5% and only 1 case greater than 5%. They show that starting from models created from small-size MKP instances with $n = 100$, it is possible to predict high-quality solutions for other problems with greater variables and constraints $m = 30$ and $n = 500$.

The GaPR improved significantly the initial solution and reached high-quality solutions. For almost all the cases, the *gap* is less than 1% and the worst fitness is close to the best fitness. That means GaPR is able to reach areas of the best solutions thanks to high-quality individuals in the initial populations. Furthermore, GaPR find better solution than the known best solutions for 3 cases. Also, GaPR even with the worst GPR solutions was able to improve their fitness.

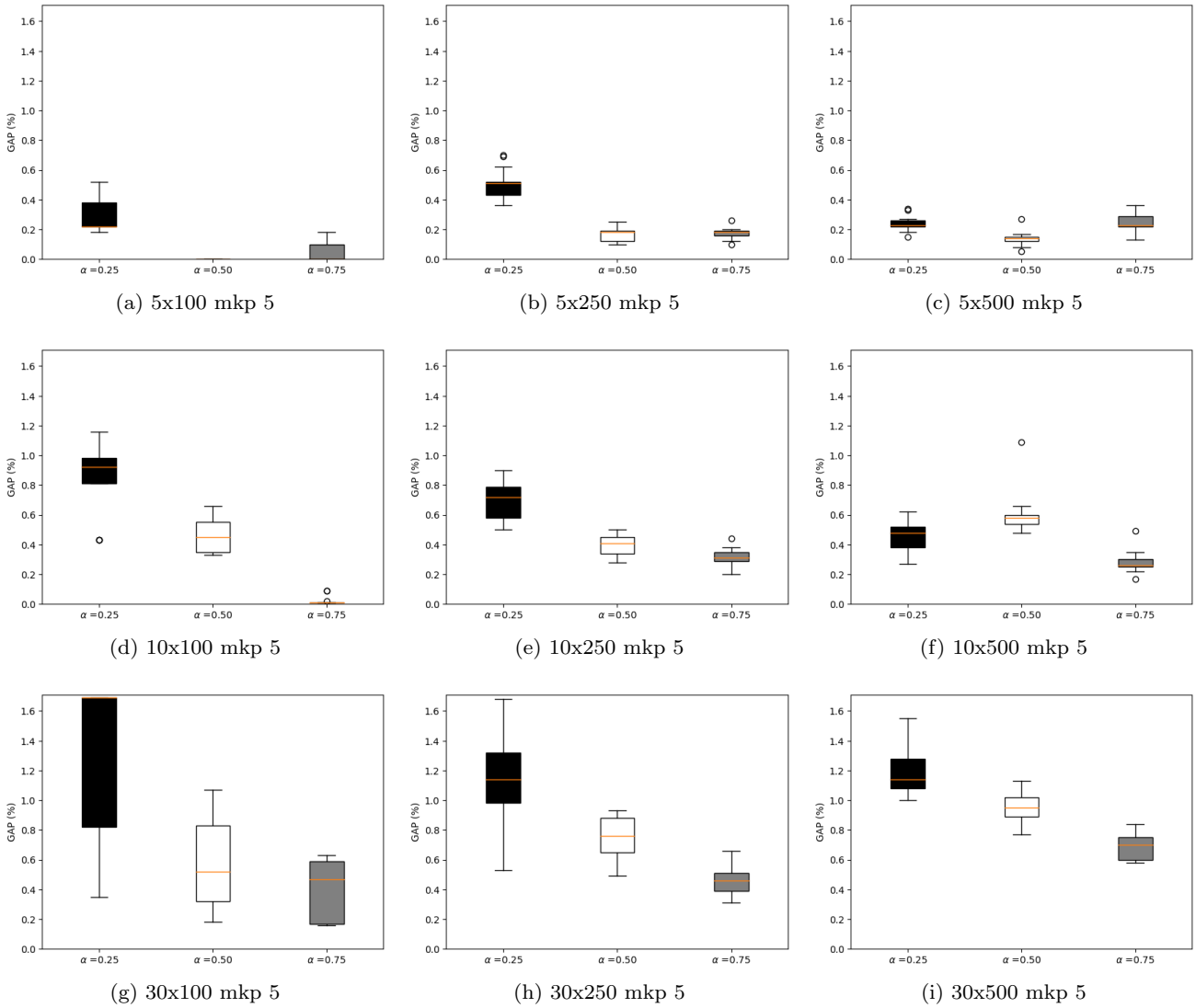
Turning now to the comparison of GaPR with the literature. Table 6 shows a comparison of GaPR to many other approaches. The proposed approach is compared to many other approaches: Shuffled Complex Evolution (SCE) [3]; general-purpose computation on graphics processing units Simulated Annealing (GPGPU SA)

m	instance	optimum	*averagef*	*fworst*	*fbest*	*PSG*	*gap*	gap best
5	0.25-1	120148	119940,3	119837	120024	1,8	0,17	0,10
	0.25-2	117879	117594,6	117450	117785	2,1	0,24	0,08
	0.25-3	121131	120797,3	120585	120905	3,7	0,27	0,18
	0.50-1	218428	217846,7	217606	218077	3,0	0,27	0,16
	0.50-2	221202	220465,3	220003	220671	4,0	0,33	0,24
	0.50-3	217542	217208,2	217108	217380	0,8	0,15	0,07
	0.75-1	295828	293471,3	291525	294204	3,1	0,79	0,55
	0.75-2	308086	305770	304759	306732	3,9	0,75	0,43
	0.75-3	299796	298010	297352	298648	2,6	0,59	0,38
10	0.25-1	117821	117452	117053	**117945**	4,6	0,31	-
	0.25-2	119249	119161	118728	**119525**	2,6	0,07	-
	0.25-3	119215	119106,6	118542	**119367**	3,9	0,09	-
	0.50-1	217377	216778,6	216256	217013	1,9	0,27	0,16
	0.50-2	219077	218364,6	217987	218907	2,8	0,32	0,08
	0.50-3	217847	217261,2	217053	217420	2,8	0,26	0,19
	0.75-1	304387	302250,3	301817	302855	2,5	0,71	0,50
	0.75-2	302379	299757,7	298898	300257	2,8	0,86	0,70
	0.75-3	302417	300335,8	299814	300949	2,8	0,69	0,48
30	0.25-1	116056	114478,9	114035	114760	5,6	1,36	1,11
	0.25-2	114810	113968,4	113799	114263	4,7	0,73	0,47
	0.25-3	116741	115394	115176	115796	3,9	1,15	0,81
	0.50-1	218104	216225,4	215823	216539	2,6	0,86	0,72
	0.50-2	214648	212852,9	212525	213051	2,9	0,83	0,74
	0.50-3	215978	214861,5	214578	215173	2,2	0,51	0,37
	0.75-1	301675	298312,9	297720	299136	2,7	1,11	0,84
	0.75-2	300055	296720,5	295670	297209	3,3	1,11	0,95
	0.75-3	305087	301510,5	300856	302695	2,6	1,17	0,78

Table 5: Detailed results of the GaPR application on some large-size instances with n = 500.

n	m	α	ACO	MCF	SCE	GPGPU SA	Sequential SA	GAGP	GaPR
5	100	0.25	0.51	1.09	3.5	0.44	1.02	0.35	**0.09**
		0.50	0.19	0.57	2.6	0.21	0.55	0.48	**0.08**
		0.75	0.09	0.38	1.1	0.14	0.28	0.21	**0.05**
	250	0.25	0.44	0.41	4.3	0.28	0.67	0.58	**0.22**
		0.50	0.19	0.22	3.3	**0.12**	0.28	0.36	0.17
		0.75	0.08	0.14	1.5	**0.06**	0.16	0.23	0.16
	500	0.25	0.27	0.21	4.6	**0.14**	0.39	0.51	0.31
		0.50	0.12	0.1	3.6	**0.06**	0.17	0.36	0.25
		0.75	0.06	0.06	1.8	**0.04**	0.11	0.22	0.76
10	100	0.25	1.83	1.87	6.8	1.27	1.94	1.0	**0.32**
		0.50	0.81	0.95	5.1	0.49	0.84	0.53	**0.17**
		0.75	0.38	0.53	2.4	0.23	0.43	0.27	**0.12**
	250	0.25	1.07	0.79	6.9	0.59	1.04	0.75	**0.44**
		0.50	0.58	0.41	5.4	0.34	0.55	0.48	**0.32**
		0.75	0.24	0.24	2.8	**0.15**	0.27	0.27	0.27
	500	0.25	0.53	0.44	6.8	**0.26**	0.57	0.71	0.64
		0.50	0.29	0.2	5.8	**0.15**	0.29	0.4	0.46
		0.75	0.16	0.13	3.4	**0.08**	0.17	0.29	0.89
30	100	0.25	3.03	3.61	8.6	1.83	2.33	1.56	**0.63**
		0.50	2.06	1.6	6.6	0.98	1.26	1.07	**0.28**
		0.75	0.93	0.97	3.6	0.67	0.73	0.36	**0.14**
	250	0.25	2.83	1.75	8.3	1.58	1.84	1.66	**0.96**
		0.50	1.56	0.79	6.9	0.78	0.82	1.0	**0.54**
		0.75	0.72	0.43	3.8	**0.38**	0.46	0.5	0.44
	500	0.25	1.98	1.05	8.6	**1.04**	1.29	4.07	1.42
		0.50	0.96	**0.44**	7.4	0.52	0.61	2.14	0.92
		0.75	0.53	**0.27**	4	0.28	0.33	0.51	1.26

Table 6: The GaPR algorithm compared to other recent results of the literature.



(a) 5x100 mkp 5 (b) 5x250 mkp 5 (c) 5x500 mkp 5

(d) 10x100 mkp 5 (e) 10x250 mkp 5 (f) 10x500 mkp 5

(g) 30x100 mkp 5 (h) 30x250 mkp 5 (i) 30x500 mkp 5

Fig. 5: Analysis of the GaPR behaviour when α, n and m changes.

and Sequential SA [8]; Ant Colony Optimisation (ACO) [11] algorithm for which the reported results are produced for the parameters of 256 ants and 8 colonies; our Genetic Algorithm Guided by Pre-treatment (GAGP) [27] and Modified Choice Function - Late Acceptance Strategy (MCF) [9]. Table 6 reports the average GAP of all the algorithms and the best result is in bold.

GaPR performed better than the other approaches for 14 of the 27 groups. Moreover, the algorithm almost reached only optimal solutions for many groups. It can be seen clearly that this algorithm has significantly improved our results produced by GAGP. In addition, GaPR are competitive with ACO and GPGPU SA both very sophisticated algorithms. GaPR are comparative to GPGPU SA for n = 100; GPGPU SA results are superior than ours for n = 250 in some cases; but our approach performed better for n = 500.

The initialisation operator based on GPR solution has lead to a population composed of individuals that contains almost the optimal solution. The individuals of the initial population requires only deep search to achieve optimal solutions. Focusing the search to the local neighbourhood, the process has more chance to reach high-quality solutions. It is possible and recommended to use small-size MKP data to build model that allows to predict helpful information to facilitate solving large-size MKP if data have similar conception. The repair procedure uses selected information combined with a small rate of random permitted to remove the worst items and to replace them by the best once.

8 Conclusion

ML and CO are two different research domains of mathematics and computer science. But many researches try to tackle CO problem using ML tools and obtain encouraging findings. The purpose of this study is to utilise ML powerful tools to deal with the multidimensional knapsack problem (MKP). Therefore, a procedure was given to explain how to build models and then to predict how likely items are to be selected. A comparison implying five ML methods have been conducted on standard MKP data set including BAR, NNR, SVR, GPR and RFR. It revealed that firstly, ML methods successfully help to produce high-quality solutions and secondly, GPR performed better than the other methods. The MKP solutions needed a close look at their neighbourhood to approach optimal solution. Therefore, a Genetic Algorithm (GA) named GaPR was proposed that includes GPR solution to initialise its population. An experimental test was conducted on well-known complex data set and has given competitive results. This work has proven that applying ML method for MKP gives encouraging results.

In optimisation situations where time is a critical factor such as real-time optimisation, it is significantly practical to adopt our approach to quickly find good configuration instead of executing time-consuming optimisation process. The main challenge to overcome is to build a complete and coherent input (feature and label). The more specific the input is, the more accurate the optimisation will be. In our study, with more features the procedure could give better outputs only by model prediction without optimisation process. Furthermore, the approach makes finding optimal solutions easier, in fact, it requires a quick and deeper look at the neighbourhood of the predicted solutions. As perspective, it is necessary to develop tools that allow to use ML method independently; reinforcement learning and graph-nature CO problems are capable to be generalised.

References

1. Mohamed Abdel-Basset, Doaa El-Shahat, Hossam Faris, and Seyedali Mirjalili. A binary multi-verse optimizer for 0-1 multidimensional knapsack problems with application in interactive multimedia systems. *Computers & Industrial Engineering*, 132:187–206, 2019.
2. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
3. Marcos Daniel Valadão Baroni and Flávio Miguel Varejão. A shuffled complex evolution algorithm for the multidimensional knapsack problem using core concept. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 2718–2723. IEEE, 2016.
4. George J Beaujon, Samuel P Marin, and Gary C McDonald. Balancing and optimizing a portfolio of r&d projects. *Naval Research Logistics (NRL)*, 48(1):18–40, 2001.
5. Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.
6. Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *arXiv preprint arXiv:1811.06128*, 2018.
7. Paul C Chu and John E Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86, 1998.
8. Bianca De Almeida Dantas and Edson Norberto Cáceres. A parallelization of a simulated annealing approach for 0-1 multidimensional knapsack problem using gpgpu. In *2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 134–140. IEEE, 2016.
9. John H Drake, Ender Özcan, and Edmund K Burke. Modified choice function heuristic selection for the multidimensional knapsack problem. In *Genetic and Evolutionary Computing*, pages 225–234. Springer, 2015.
10. Patrick Emami and Sanjay Ranka. Learning permutations with sinkhorn policy gradient. *arXiv preprint arXiv:1805.07010*, 2018.
11. Henrique Fingler, Edson Norberto Cáceres, Henrique Mongelli, and Siang W Song. A cuda based solution to the multidimensional knapsack problem using the ant colony optimization. In *ICCS*, pages 84–94, 2014.
12. Siddig Jihad, Xianqiao Chen, Bing Shi, and Solyman Aiman. Multidimensional knapsack problem for resource allocation in a distributed competitive environment based on genetic algorithm. In *2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pages 1–5. IEEE, 2019.
13. Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem.

arXiv preprint arXiv:1906.01227, 2019.

14. Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.
15. Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
16. Wouter Kool, HV Hoof, and Max Welling. Attention solves your tsp, approximately. *Statistics*, 1050:22, 2018.
17. Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.
18. Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! In *Seventh International Conference on Learning Representations ICLR 2019*, 2019.
19. Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pages 539–548, 2018.
20. Michele Lombardi and Michela Milano. Boosting combinatorial problem modeling with machine learning. *arXiv preprint arXiv:1807.05517*, 2018.
21. Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *arXiv preprint arXiv:2003.03600*, 2020.
22. Helga Meier, Nicos Christofides, and Gerry Salkin. Capital budgeting under uncertainty-an integrated approach using contingent claims analysis and integer programming. *Operations Research*, 49(2):196–206, 2001.
23. Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.
24. Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3303–3313, 2018.
25. Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9839–9849, 2018.
26. Jakob Puchinger, Günther R Raidl, and Ulrich Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22(2):250–265, 2010.
27. Abdellah Rezoug, Mohamed Bader-El-Den, and Dalila Boughaci. Knowledge-based genetic algorithm for the 0–1 multidimensional knapsack problem. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2030–2037. IEEE, 2017.
28. Shizuo Senju and Yoshiaki Toyoda. An approach to linear programming with 0-1 variables. *Management Science*, pages B196–B207, 1968.
29. El-Ghazali Talbi. Machine learning into metaheuristics: A survey and taxonomy of data-driven metaheuristics. 2020.
30. Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700, 2015.