International Journal of Machine Learning and Cybernetics
https://doi.org/10.1007/s13042-020-01085-8

ORIGINAL ARTICLE

Enhancing a machine learning binarization framework by perturbation operators: analysis on the multidimensional knapsack problem

José García[1] · Eduardo Lalla-Ruiz[2] · Stefan Voß[3] · Enrique López Droguett[4]

Received: 5 January 2019 / Accepted: 8 February 2020

Abstract
Solving combinatorial optimization problems is of great interest in the areas of computer science and operations research. Optimization algorithms and particularly metaheuristics are constantly improved in order to reduce execution times, increase the quality of solutions and address larger instances. In this work, an improvement of the binarization framework which uses the K-means technique is developed. To achieve this, a perturbation operator based on the K-nearest neighbor technique is incorporated into the framework with the aim of generating more robust binarized algorithms. The technique of K-nearest neighbors is used for improving the properties of diversification and intensification of metaheuristics in its binary version. The contribution of the K-nearest neighbors perturbation operator to the final results is systematically analyzed. Particle Swarm Optimization and Cuckoo Search are used as metaheuristic techniques. To verify the results, the well-known multidimensional knapsack problem is tackled. A computational comparison is made with the state-of-the-art of metaheuristic techniques that use general mechanisms of binarization. The results show that our improved framework produces consistently better results. In this sense, the contribution of the operator which uses the K-nearest neighbors technique is investigated finding that this operator contributes significantly to the quality of the results.

Keywords Combinatorial Optimisation · Machine Learning · Metaheuristics · KNN · K-means · Knapsack

1 Introduction

Decision making in complex systems is a cross-cutting activity in different areas of engineering and management. Many of these decisions require evaluating a very large combination of elements as well as having to solve a combinatorial optimization problem (COP) to find a feasible and satisfactory result. Examples of COP are found in the areas of logistics Korkmaz et al. [32], transportation García et al. [25], machine learning Al-Madi et al. [2] biology Guo et al. [27], and many others. Depending on the problem definition, many COPs can be categorized as $\mathcal{NP}$-hard. Among the most successful ways to address such problems, a common solving way is to simplify the model in order to try to solve instances of small to medium size through exact techniques, or addressing them through heuristic or metaheuristic algorithms. This last solving option allows to deal with large-size problems, but without ensuring the optimality of the solutions. Therefore, research lines that allow obtaining robust algorithms associated with the solution of COPs are

✉ José García
jose.garcia@pucv.cl

Eduardo Lalla-Ruiz
e.a.lalla@utwente.nl

Stefan Voß
stefan.voss@uni-hamburg.de

Enrique López Droguett
elopezdroguett@ing.uchile.cl

[1] Pontificia Universidad Católica de Valparaíso, 2362807 Valparaíso, Chile

[2] Department of Industrial Engineering and Business Information Systems, Universiteit Twente, Enschede, The Netherlands

[3] Institute of Information Systems, University of Hamburg, Hamburg, Germany

[4] Department of Mechanical Engineering, University of Chile, Beauchef 850, Santiago, Chile

Published online: 21 February 2020

of great interest in the areas of computer science and operations research.

During the last decade, important investigation lines have explored the hybridization among different optimization techniques with the goal of obtaining more robust methods in terms of quality of solutions and convergence times. In the literature, the main hybridization proposals are: (i) *matheuristics*, which combine heuristics or metaheuristics with mathematical programming Caserta and Voß [6], (ii) *hybrid heuristics*, the combination between different heuristic or metaheuristic methods Talbi [51], (iii) *simheuristics*, that combine simulation and metaheuristics Juan et al. [30], and (iv) the hybridization between metaheuristics and machine learning. The latter hybridization field between machine learning and metaheuristics is an emerging research line in the area of operations research. In this sense, we find that the combination can occur such that metaheuristics help machine learning algorithms to improve their results (e.g., [9, 62]) or in the opposite direction where machine learning techniques help in the robustness of metaheuristic algorithms (e.g., [16]). The details of the hybridization forms are specified in Sect. 2.1.

In this article, inspired by the mentioned research lines, we explore the application of an automatic learning algorithm in a perturbation operator to improve the diversification and intensification properties of a given metaheuristic when addressing combinatorial optimization problems. The contributions of this work are detailed below:

- An improvement of the automatic learning binarization framework designed in García et al. [24] is proposed in order to allow metaheuristics commonly defined and used in continuous optimization addressing COP efficiently. This framework uses the k-means unsupervised learning technique to perform the binarization process and in this article, the nearest K-neighbors technique is included to improve the diversification and intensification properties of a given metaheuristic. The selected metaheuristics are particle swarm optimization (PSO) and cuckoo search (CS). Their selection is based on the fact that they are commonly used in continuous optimization and allow an easy way to adjust their parameters in continuous spaces. In this sense, theoretical models of PSO convergence already exist.
- These hybrid metaheuristics are applied to the well-known multidimensional knapsack problem (MKP). This problem has been extensively studied, and because of that, small, medium, and large instances are available in the literature. We have utilized the large instances as the ones used to evaluate the contribution of the KNN perturbation operator. On the other hand, the MKP has numerous practical real-world applications, such as computer scheduling programs in multiprogramming environments, allocation of shelf space to a consumer product in retail stores, the capital-budgeting problem, among many others.
- For a proper evaluation of our hybrid algorithms first, we use a parameter estimation method developed in García et al. [24] to determine the best metaheuristic configurations. Subsequently, experiments are developed with the aim of shedding light on the contribution of the KNN operator in the framework. Finally, our hybrid algorithms are compared with the latest generation binarization frameworks. For this purpose, we use the larger problems of the OR-Library[1]. The numerical results show that our hybrid algorithms achieve highly competitive results.

The rest of the article is structured as follows. Section 3 describes the MKP and some of its applications. In Sect. 2, a state-of-the-art hybridization between the areas of machine learning and metaheuristics is provided and main binarization methods are described. Later, in Sect. 4, the proposed hybrid algorithm is detailed. The results of the contribution of the KNN operator are provided in Sect. 5. To evaluate the quality of our results, in Sect. 6 we provide a comparison with algorithms that use generic binarization mechanisms. Finally, in Sect. 7 conclusions and some future lines of research are given.

2 Related works

2.1 Hybridizing metaheuristics with machine learning

When it comes to integrating machine learning and metaheuristics, two large groups can be mainly indicated. The first group corresponds to metaheuristic techniques for improving the performance of machine learning algorithms. The second covers machine learning algorithms for enhancing metaheuristics performance. For the first group, we find four main areas of application: improving clustering algorithms, feature selection applications, improving classification algorithms and strengthening regression algorithms. The summary of the integration between these two areas is shown in Fig. 1.

In the case of clustering algorithms, a variety of methods has been reported. One of the main problems that presents a greater algorithmic complexity, corresponds to the search of centroids for grouping in a better way the set of studied objects. Since this problem is $\mathcal{NP}$-hard, approximate methods have been proposed to address it. In this regard, there

[1] OR-Library: http://www.brunel.ac.uk/mastjjb/jeb/orlib/mknapinfo.html.

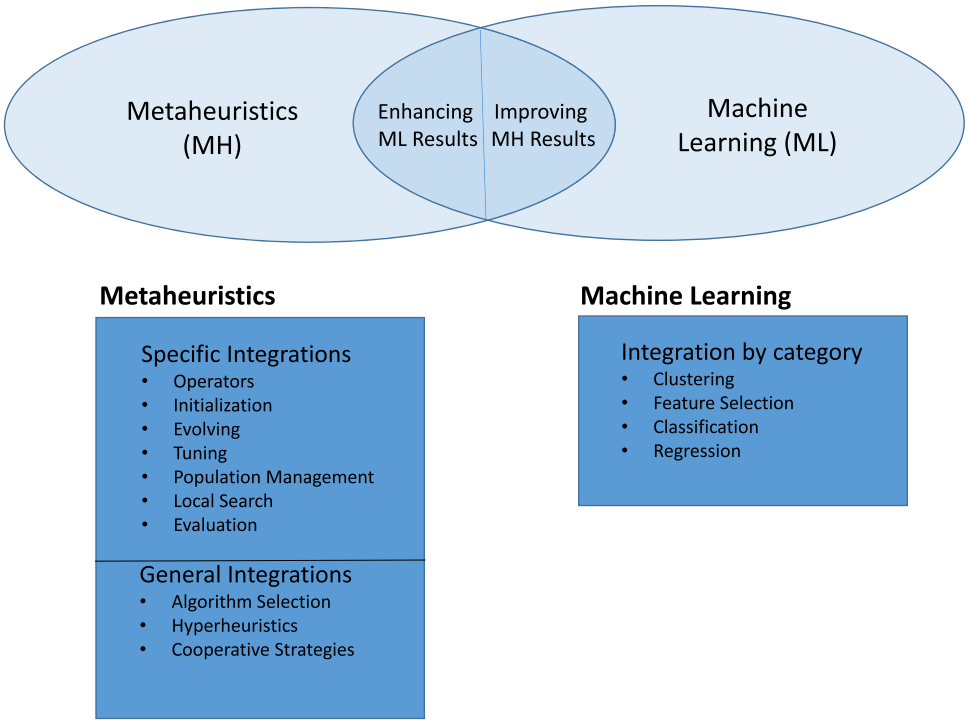

Fig. 1 General scheme: combining machine learning and metaheuristics. Adapted and extended from Calvet et al. [5]

is a long list of studies in this area, however, in recent years the focus has moved to solving applied problems. In Mann and Singh [40], an improved artificial bee colony algorithm was applied to solve the problem of energy-efficient clustering in a wireless sensor network. In Mirghasemi et al. [42] we found the use of particle swarm optimization and fuzzy C-means to perform segmentation of images with noise. The planning of helicopter transportation of employees to the production platforms of oil and gas was studied in de Alvarenga Rosa et al. [14], using a cluster search metaheuristic. In that work, the metaheuristic approach was compared with a general-purpose solver (i.e., CPLEX), indicating that the former produces better, more stable solutions in shorter computational time. Another interesting application in logistics management corresponds to the management policy of warehousing and item assignment. In Kuo et al. [33], a PSO algorithm was applied for the item assignation problem in a synchronized zone order picking system. Finally, in Kuo et al. [34] a clustering method based on metaheuristics was proposed to solve a client segmentation problem.

A major difficulty in the learning process of a machine learning algorithm is related to the dimension of the dataset. The inadequate handling of the dimension of the dataset can involve problems such as under or over-fitting plus large amounts of computations necessary for its training. Because of its definition, feature selection is a combinatorial problem and has been effectively addressed by metaheuristic algorithms. In Ahmad et al. [1], metaheuristic algorithms were compared with traditional feature selection methods, applying them to datasets used in sentiment analysis. The selection of features is fundamental in real-time data stream mining problems; e.g., in Fong et al. [18] an accelerated PSO was proposed to efficiently address feature selection. A self-adaptive PSO was developed in Xue et al. [56] to address the large-scale feature selection problem in classification.

The problems of classification and regression form an important group of problems that are usually addressed through supervised learning techniques. The contribution of metaheuristics in supervised learning algorithms in both classification and regression problems has been significant. Metaheuristics have contributed to the improvement of algorithms such as support vector machine, artificial neural networks, decision trees, logistic regression, among others. In Chou and Thedja [11], a classification system was proposed that integrates a firefly algorithm with the least squares support vector machine technique to apply it to geotechnical problems. Classification systems applied to healthcare using metaheuristic algorithms and big data techniques are detailed in Tsai et al. [52]. In Fernandes et al. [17], metaheuristics were used to design an enhanced probabilistic neural network algorithm. In regression problems, for example, by applying metaheuristics in time series using sliding-window in Chou and Nguyen [9], they design a model to predict the stock prices of Taiwan's construction companies. In Chou and Pham [10], by using the firefly algorithm, the parameters of least squares support vector regression are optimized for enhancing prediction accuracy in engineering design. The shear strength prediction in reinforced concrete

deep beams was addressed in Chou et al. [8]. In that article, they propose a firefly algorithm integrated with a support vector machine algorithm for accurately predicting shear strength.

On the other hand, the contribution of machine learning techniques to strengthen metaheuristic algorithms has been important. In this case, we distinguish two large groups according to the way they are integrated. A first group corresponds to specific integrations where the techniques of machine learning are inserted through an operator into one of the metaheuristic modules. The second group corresponds to general integrations where the machine learning technique works as a selector of different metaheuristic algorithms, choosing the most appropriate one for each instance.

In the case of a specific integration, we find integrations in different modules of metaheuristics. For example, during the initialization of solutions, performing the tuning of parameters, in the binarization of continuous metaheuristics, also machine learning is used in the population management, etc. In the tuning of parameters, in De Jong [15] the author applies a dynamic tuning approach for adapting the parameters depending on the instance and the evolution of the algorithm. A chess rating method was applied in Veček et al. [55]. That method was compared with other techniques such as f-race showing a good performance. In Ries and Beullens [45], a decision tree was used to perform the tuning of the parameters. Usually, the mechanism of solution initialization of a metaheuristic is done in a random way or using some heuristic. However, there are attempts to use machine learning in the initialization of solutions. In Li et al. [36], a case-based reasoning was used to initialize a genetic algorithm and apply it to the weighted circles layout problem. Hopfield neural networks were used in Yalcinoz and Altun [57] to initiate solutions of a genetic algorithm that were used to solve the economic dispatch problem. With regards to population management, the main line of research is related to extracting information from the solutions previously visited in the search space and identifying the regions that have the greatest potential to exploit them. In the literature, we can observe the use of clustering techniques to improve the exploration of the search space Streichert et al. [49]. Also, the use of case-based reasoning techniques was investigated in Santos et al. [46] in order to identify subspaces of searches to solve the single-vehicle routing problem. In Jin et al. [29], an incremental learning technique was used to apply this to the constrained portfolio optimization problem. Finally, a research area with a lot of activity corresponds to designing binary versions of algorithms that work naturally in continuous spaces to enable them to work on combinatorial problems. In this area, the application of clustering techniques in García et al. [20] to perform the binarization has been proposed. In García et al. [21, 22], the percentile concept and the ranking of the solutions were used to obtain binary algorithms from continuous algorithms. In García et al. [20] the distributed computing framework Apache Spark was applied to generate distributed versions of the binarized algorithms.

Concerning general integrations, we can point out three main groups: algorithm selection, hyperheuristics, and cooperative strategies. In the case of algorithm selection, the objective is to choose from a portfolio of algorithms together with a set of characteristics associated with each instance of the problem. In Smith-Miles et al. [48] meta-learning techniques were used with the goal of proposing a methodology to measure the strengths and weaknesses of algorithms that solve optimization problems in different instances. The berth scheduling problem at bulk terminals was addressed in de León et al. [16] by using a machine-learning-based algorithm selection approach. The goal of hyperheuristics is to automate the design of heuristic or metaheuristic methods to address a wide range of problems. In Tyasnurita et al. [53], an artificial neural network was used to improve the performance of hyperheuristics when solving different instances of the vehicle routing problem. The problem of nurse rostering was addressed in Asta et al. [3] through a tensor-based hyperheuristic algorithm. Finally, in Damaševičius and Woźniak [13] a hyperheuristic was designed which allows integrating different nature-inspired algorithms. Cooperative strategies consist of combining algorithms in a parallel or sequential way in order to obtain more robust methods. Cooperation can be completed by sharing a complete solution or part of it. In Cadenas et al. [4], a centralized cooperative strategy was developed where knowledge was modeled through fuzzy rules. In Martin et al. [41], a distributed framework based on agents was proposed. Each agent corresponds to a metaheuristic, where the agent has the ability to adapt through direct cooperation. This framework was applied to the permutation flow shop problem.

2.2 Binarization methods

Nowadays, there is a series of metaheuristic algorithms that are designed to work in continuous spaces. Among the most prominent, Particle Swarm Optimization (PSO) and Cuckoo Search (CS) can be marked as some of the most used ones. On the other hand, the existence of a large number of $\mathcal{NP}$ -hard combinatorial problems motivates the investigation of robust mechanisms that allow adapting these continuous algorithms to discrete versions.

In a review of the state-of-the-art of binarization techniques Crawford et al. [12], two approximations were identified. A first approach considers general methods of binarization. In those general methods, there is a mechanism that allows transforming any continuous metaheuristic into a binary one without altering the metaheuristic operators. In this approach, the main frameworks used are transfer

functions and angle modulation. The second approach corresponds to binarizations where the way of operating metaheuristics is specifically altered. Within this second approach, techniques such as Quantum binary and Set-based approaches can be highlighted.

Transfer functions: The simplest and most widely used binarization method corresponds to the transfer functions. The transfer functions were introduced by Kennedy and Eberhart [31] to generate binary versions of PSO. PSO considers each solution as a particle. This particle has a position given by a solution for some iteration and a velocity which corresponds to the vector obtained from the difference of the particle position in two consecutive iterations. The transfer function is a very simple operator and relates the velocity of the particles in PSO with a transition probability. The transfer function takes values from $\mathbb{R}^n$ and generates transition probability values in $[0, 1]^n$. The transfer functions force the particles to move in a binary space. Depending on its shape, these are usually classified as *S*-Shape Yang et al. [59] and *V*-Shape García et al. [23]. Once the function produces the value between 0 and 1, the next step is to use a rule that allows obtaining 0 or 1. For this, well-defined rules are applied that use the concepts of complement, elite, random, among others.

Angle Modulation: This method is based on the family of trigonometric functions shown in Eq. (1). These functions have four parameters responsible for controlling the frequency and displacement of the trigonometric function.

$$g_i(x_j) = \sin(2\pi(x_j - a_i)b_i \cos(2\pi(x_j - a_i)c_i)) + d_i \tag{1}$$

The first time this method was applied to binarizations was in PSO. In this case, the binary PSO was applied to benchmark functions. Assume a given binary problem of dimension n and let $X = (x_1, x_2, \ldots, x_n)$ be a solution. We start with a four-dimensional search space. Each dimension represents a coefficient of Eq. (1). Then every solution (a_i, b_i, c_i, d_i) is associated to a g_i trigonometric function. For each element x_j the following rule is applied:

$$b_{ij} = \begin{cases} 1 & \text{if } g_i(x_j) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Then, for each initial solution of 4 dimensions (a_i, b_i, c_i, d_i), function g_i which is shown in Eq. (1) is applied and then Eq. (11) is utilized. As a result, a binary solution of dimension n $(b_{i1}, b_{i2}, \ldots, b_{in})$ is obtained. This is a feasible solution for our n-binary problem. The angle modulation method has been applied to network reconfiguration problems Liu et al. [38] using a binary PSO method, to the antenna position problem using an angle modulation binary bat algorithm Moiz et al. [43], and to a multi-user detection technique Swagatam et al. [50] using a binary adaptive evolutionary algorithm.

Quantum binary approach: Considering the line of research that integrates the areas of Evolutionary computation (EC) and Quantum computation, there are mainly three categories of algorithms Zhang [60].

1. Quantum evolutionary algorithms: Corresponds to the design of EC algorithms to be applied in a quantum computing environment.
2. Evolutionary-designed quantum algorithms: These algorithms try to automate the generation of new quantum algorithms using Evolutionary algorithms.
3. Quantum-inspired evolutionary algorithms: This category uses quantum computing concepts to strengthen EC algorithms.

In particular, the Quantum binary approach belongs to Quantum-inspired evolutionary algorithms. Specifically, this approach adapts the concepts of q-bits and superposition used in quantum computing to work in normal computers.

In the Quantum binary approach method, each feasible solution has a position $X = (x_1, x_2, \ldots, x_n)$ and a quantum q-bits vector $Q = [Q_1, Q_2, \ldots, Q_n]$. Q represents the probability of x_j taking the value 1. For each dimension j, a random number between [0,1] is generated and compared with Q_j, if $rand < Q_j$, then $x_j = 1$, else $x_j = 0$. The upgrade mechanism of the Q vector is specific to each metaheuristic.

The main difficulty that general binarization frameworks have is related to the concept of Spatial disconnect Leonard et al. [35]. Spatial disconnect is originated when nearby solutions generated by metaheuristics in the continuous space are not transformed into nearby solutions when applying the binarization process. Roughly speaking, we can think of a loss of the continuity of the framework. This phenomenon of Spatial disconnect has the consequence that the properties of exploration and exploitation are altered and, therefore, the precision and convergence of metaheuristics worsen. A study was developed on how the transfer functions affect the exploration and exploitation properties in Saremi et al. [47]. For angle modulation, the study was developed in Leonard et al. [35].

On the other hand, specific binarization algorithms, that modify the operators of the metaheuristic, are susceptible to problems such as Hamming cliffs, loss of precision, search space discretization and the curse of dimensionality Leonard et al. [35]. This was studied by Pampara [44] and for the particular case of PSO by Chen et al. [7]. In the latter, the authors observed that the parameters of the Binary PSO change the speed behavior of the original metaheuristic.

3 Applications of the multidimensional knapsack problem

The multidimensional knapsack problem (MKP, [19]), is a non-deterministic polynomial-time ($\mathcal{NP}$)-hard combinatorial problem that considers multiple resource constraints, Garey and Johnson [26] Its goal is to fill a given multidimensional capacity-limited knapsack with a subset of items in order to get the maximum benefit associated with the profit of each selected item. The selection of items has to consider the limitations of resource capacity since each element has different resource requirements. Formally, the problem is defined as follows.

$$\text{Maximize } P(x_1, \ldots, x_n) = \sum_{j=1}^{n} p_j x_j. \tag{3}$$

subject to:

$$\sum_{j=1}^{n} c_{ij} x_j \leq b_i, i \in \{1, \ldots, m\}. \tag{4}$$

$$x_{ij} \in \{0, 1\}. \tag{5}$$

where b_i corresponds to the capacity limitation of resource $i \in M$. Each element $j \in N$ has a requirement of c_{ij} regarding resource i as well as a benefit p_j. Moreover, $x_j \in \{0, 1\}$ indicates whether the element is in the knapsack or not, $j \in \{1, \ldots, n\}$, $c_{ij} \geq 0$, $p_j > 0$, $b_j > 0$, n corresponds to the number of items, and m the number of knapsack constraints.

In the rest of this section, with the aim to show the importance of solving the MKP, we detail some MKP applications in the real world. The first problem to be detailed corresponds to the daily photographic scheduling (DPS) problem of an earth observation satellite Vasquez and Hao [54]. Let $F = (f_1, \ldots, f_m)$ be a set of candidate photographs to be taken the next day by the satellite. Since the satellite has more than one camera, the cameras are denoted by $C = (c_1, \ldots, c_k)$. In order to represent the problem in a binary form, it is necessary to define the vector $X = (x_1, \ldots, x_n)$, where x_1 represents the pair (f_1, c_1), $x_2 = (f_1, c_2)$, and so on. Each pair $x_j =$ (photograph, camera), associates a profit p_j which results from an aggregation of a series of conditions such as the urgency of the photograph, the importance of the client, meteorological forecast, etc. In addition, the set of photos, associates a set of hard constraints related to the storage capacity, restrictions associated with non-overlapping pictures and constraints associated with the instantaneous data flow.

The DPS problem model associated with a satellite with three mono and one stereo cameras is modeled by expressions (6) to (9):

$$\text{Maximize } P(x_1, \ldots, x_n) = \sum_{j=1}^{n} p_j x_j \tag{6}$$

where $p_j \in \mathbb{Z}^+$. and subject to:

$$\sum_{j=1}^{n} c_j x_j \leq C_1 \tag{7}$$

$$x_i + x_j \leq 1, \forall (x_i, x_j) \in C_2 \tag{8}$$

$$x_i + x_j + x_k \leq B, \forall (x_i, x_j, x_k) \in C_3 \tag{9}$$

$$x_{ij} \in \{0, 1\} \tag{10}$$

where C_1 represents the capacity of the knapsack, C_2 indicates that each photo is taken by a single camera, and C_3 models flow and overlay constraints, $B \in \{1, 2\}$.

Another interesting problem related to the MKP is the shelf space allocation problem (SSAP) Yang [58]. The idea is to manage the shelf space through intelligent systems. Properly solving this problem not only decreases the level of inventories, but also improves the relationship between the seller and the customer. When we consider the SSAP, the objective function corresponds to the profit of all products of the store. Additionally, the problem has constraints associated with the store's total storage capacity along with maintaining an adequate product mix. Formally, the problem is described as follows.

$$\text{Maximize } P(x_1, \ldots, x_n) = \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} x_{ij} \tag{11}$$

subject to:

$$\sum_{i=1}^{n} a_i x_{ij} \leq T_j, j = \{1, \ldots, m\} \tag{12}$$

$$L_i \leq \sum_{j=1}^{m} x_{ij} \leq U_i, i = \{1, \ldots, n\} \tag{13}$$

$$x_{ij} \in \{\mathbb{Z}_0^+\} \tag{14}$$

A store has n product items determined by a product mix decision and these items can be displayed on m shelves. Then, x_{ij} represents the amount of product i that can be located on the shelf j. The length of shelf j is T_j and each facing of product i (interpreted as the width of showing i once on the shelf), is a_i long. To maintain the stock balance of each product, lower and upper limits are defined. The lower limit L_i associated with a product i, is necessary to guarantee that customers still find enough units of item i.

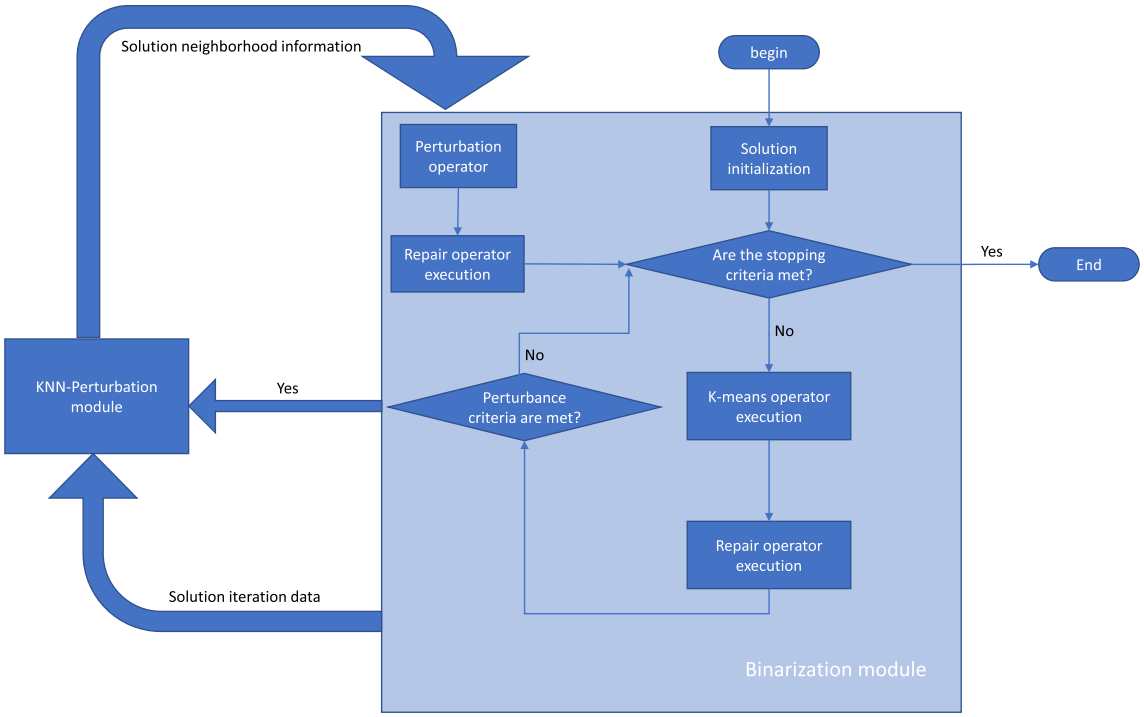

Fig. 2 Flow chart of machine learning swarm intelligence algorithm

The upper limit U_i of a product i, ensures that an adequate space is left for other products.

4 Hybrid algorithm

In this section, the algorithm used to solve the MKP is detailed. Our algorithm is composed of two modules: The KNN perturbation module diagrammed in the left part of Fig. 2 and the binarization module shown in the right part of Fig. 2. The KNN perturbation module uses the KNN algorithm in order to obtain the nearest neighbors of a particular solution. This module feeds on a subset of data generated in the different iterations of the binarization module. The aim of the binarization module is to binarize the solutions generated by continuous swarm intelligence algorithms such as PSO and CS. The binarization module uses four operators: The solutions initialization operator described in Sect. 4.2, the K-means operator detailed in Sect. 4.3, a repair operator described in Sect. 4.5, and the KNN perturbation operator detailed in Sect. 4.4.

4.1 KNN perturbation module

In this section, the operation of the KNN perturbation module is explained in detail. The final goal of this module is to calculate, using a measure, the importance of each dimension in the neighbourhood of the solution when trying to solve a particular instance of the MKP. As a measure to estimate importance, a definition similar to the Morris method used in Iooss and Lemaître [28] is used. The objective of the Morris method is to determine the input variables that are most important using a small number of evaluations to perform the calculation. The method allows determining linear influences, non-linear influences, and interactions between the variables.

Consider N points in the search space given by $S = \{X^j / j \in \{1, \ldots, N\}\}$, then we define $EE_i(X), X \in S$ by:

$$EE_i(X) = f(X_1, \ldots, \hat{X}_i, \ldots, X_d) - f(X) \tag{15}$$

where $\hat{X}_i$ corresponds to the complement, that is, if $X_i = 0$ then $\hat{X}_i = 1$. The previous calculation is performed for each of the dimensions of all solutions in S. Then, we calculate the statistical indicators average (μ_i) and standard deviation (σ_i) defined in the Eqs. 16 and 17, respectively.

$$\mu_i = \frac{1}{N} \sum_{j=1}^{N} |EE_i(X^j)| \tag{16}$$

$$\sigma_i = \sqrt{\frac{1}{N} \sum_{j=1}^{N} (EE_i(X^j) - u_i)^2} \tag{17}$$

Then the pair (μ, σ) has quite clear interpretations depending on the combination of:

1. Small values of μ and σ means the variable has a small effect on the objective function.
2. Small values of μ and high values of σ implies that the input variable has important non-linear effects.
3. High values of μ and small values of σ implies that the input variable has important linear effects.
4. High values of μ and high values of σ implies that the input variable has important non-linear effects or interaction with other variables.

The previously detailed calculations allow us to evaluate the exploration of the whole space. However, our objective is to evaluate the exploitation capability in the region of a defined solution. Therefore, the previous calculations must be adapted with the aim of incorporating the concept of the neighbourhood into the calculation. This way, to introduce that, we use the K-nearest neighbourhood algorithm. This algorithm allows us to efficiently obtain the neighbour of a solution. The calculation of the mean and the standard deviation is made using exclusively the neighbours of the solution. As a starting point, the KNN perturbation module is fed with a percentage of the solutions generated by the binarization module in each iteration. This is shown in Fig. 2 in the *Solution iteration data* arrow. In our specific case, the percentage of solutions was 25% of the best solutions obtained in each iteration. The idea of incorporating 25% of the best solutions aims to use elements which belong to the first quartile to determine with this information where the solution should be perturbed. Subsequently, each time the binarization module needs to execute a perturbation operator, it will deliver to the perturbation module a list of solutions so that the perturbation module calculates, for each solution, the measure that allows the perturbation to be made. Finally, the list of measurements is delivered to the perturbation operator which is responsible for executing the perturbation. This is shown in Fig. 2 on the *Solution neighbourhood information* arrow.

Finally, to perform the calculation of the measure, we consider the value w_i of Eq. (18), where μ_i^* and σ_i^*, correspond to normalized values of the mean and deviation, respectively. The $\sqrt{2}$ value is added to the w_i values to normalize them between 0 and 1. The details of the calculation of w for each solution are shown in Algorithm 1.

$$w_i = \frac{\sqrt{\mu_i^* + \sigma_i^*}}{\sqrt{2}} \tag{18}$$

As input, the algorithm uses the solution (*sol*) in order to obtain its corresponding weights. As an output, the algorithm delivers the weights (*ListWeight*) associated with the solution. The first function to execute corresponds to getting the K neighbours which are stored in *neighbours*. Subsequently, with the neighbour list, w_i is calculated for each dimension according to Eqs. 16–18. D represents the dimension of *sol*.

Algorithm 1 Measurement algorithm

```
Function measure(sol)
Input sol
Output The list of weights (ListWeight)
ListWeight ← []
neigbours ← getKneighbours(sol)
for (each dimension i in sol) do
    w_i ← getmeasure(neighbours)
    ListWeight.append(w_i)
end for
return ListWeight
```

4.2 Initialization operator

In the initialization of the solutions, the heuristic shown in Eq. (19) is used. In this equation c_{ij} represents the cost of the object i in the knapsack j, b_j corresponds to the capacity constraint of the knapsack j and p_i corresponds to the profit of the i element. This heuristic was proposed in García et al. [24] and its objective is to select the elements that enter the knapsack. The construction of a solution starts with the random selection of a first element, later it is consulted if it is possible to add new elements. If it is possible, from the list of elements that satisfy the constraints, the one with the best value according to Eq. 19 is selected. The process of incorporating elements continues until there are no elements that satisfy the constraints. The pseudo-code is shown in Algorithm 2.

$$\delta_i = \frac{\sum_{j=1}^{m} \frac{c_{ij}}{m(b_j - \sum_{i \in S} c_{ij})}}{p_i} \tag{19}$$

Algorithm 2 Initialization algorithm

```
Function initAlgorithm(listElements)
Input The list of elements (listElements)
Output The solution (sol)
sol ← []
element ← getRandomElement(listElements)
sol.append(element)
while (there are elements that satisfy the constraints:) do
    listPosibleElements ← getListPosibleElements(listElements)
    element ← getBestElement(listPosibleElements)
    sol.append(element)
end while
return sol
```

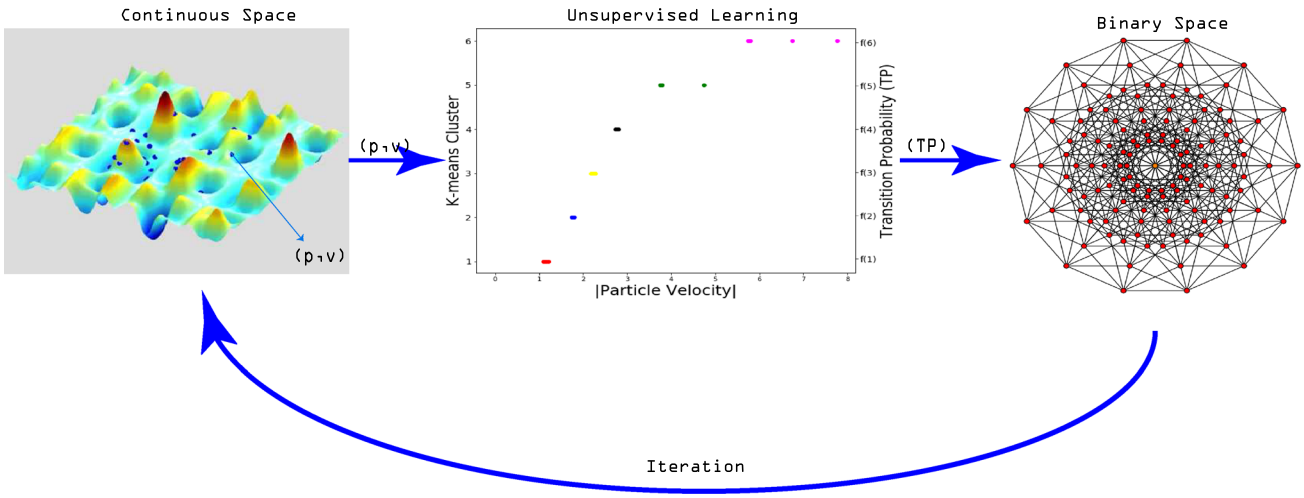

Fig. 3 K-means binarization method

4.3 K-means operator

Since PSO and CS are continuous swarm intelligence algorithms and the MKP is a combinatorial problem, a discretization framework is required. In the literature, two large groups of binarization algorithms have been proposed. The first group corresponds to specific adaptations of the continuous algorithm that can hardly be applied to other continuous algorithms. The second group corresponds to general binarization methods that usually have lower performance than the specific methods but allow adapting any continuous algorithm. For a detailed review of the different techniques, the reader is referred to Crawford et al. [12].

In order to binarize more than one continuous algorithm, in this article we use a general method of binarization based on the k-means unsupervised learning technique. This technique was chosen because it showed for the knapsack problem to be more robust than other general binarization techniques such as transfer functions García et al. [24]. In Fig. 3 the main stages of the k-means binarization method are considered. The binarization process begins with the value of the solutions (s) and their displacements (d) generated by the continuous algorithm. This stage is illustrated in the left part of Fig. 3. Because the algorithm is continuous, d takes values in $\mathbb{R}^n$ space. In the Eq. 20, the update of the solution is presented in a general way. The s_{t+1} variable represents the s solution of the particle at time t+1. This solution is obtained from the solution s at time t plus a d function calculated at time t+1. The function d is specific to each metaheuristic and produces values in $\mathbb{R}^n$. For example in Cuckoo Search $d = \alpha \oplus Levy(\lambda)(x)$, and in the PSO algorithm d can be written in a simplified form as $d = v(x)$.

$$s_{t+1} = s_t + d_{t+1}(s(t)) \tag{20}$$

Later we used d to perform the binarization. Let $d^i(s(t))$ the magnitude of the displacement $d(s(t))$ in the i-th coordinate of the solution. Then these displacements are grouped using the magnitude of the displacement $d^i(s(t))$. This grouping is done using the k-means technique where k represents the number of clusters used. This is visualized in the center of Fig. 3. Later, for each cluster, we associate a transition probability f(k) shown on the right axis of the middle chart of Fig. 3. Finally, with the value of this probability together with Eq. 21, we proceed to make the transitions in the binary space. This last part corresponds to the diagram on the right in Fig. 3.

$$x^i(t+1) := \begin{cases} \hat{x}^i(t), & \text{if } r \text{ and} < TP(x^i) \\ x^i(t), & \text{otherwise} \end{cases} \tag{21}$$

4.4 KNN-perturbation operator

A first operator considered a probability of transition of 0.3 and a second operator a probability of 0.5. The 0.3 value was used based on the result of a previous work García et al. [24]. This value was the one performing the best. In the case of 0.5, it was used because it is a random operator giving same probability of staying or making the transition.

The KNN perturbation operator is executed when the perturbation criterion depicted in Fig. 1 is met. The criterion evaluates the number of iterations in which the best solution found has not changed. If the iteration threshold is exceeded, then the operator is activated. For the implementation described in this article, the threshold of iterations corresponded to 30. The KNN perturbation operator has as

parameters: the list of current solutions and a value of the force of the perturbation η. As an output, it gives us the list of perturbed solutions.

The ϵ percentage of the best solutions are sent to the KNN perturbation module to calculate the weights that are used to perform the perturbation. Later, with the weights in each one of the variables, we proceed to perform the perturbation of the best solutions. The rest of the solutions are subjected to random perturbations handled by the indicator η. This was designed thinking that solutions that did not have good results were perturbed in a random way to adequately handle the diversification of the solutions. Finally, once the solution is perturbed, a repair operator is executed in case a restriction is violated or new elements can be incorporated. The pseudocode of the KNN perturbation operator is shown in Algorithm 3.

Algorithm 3 KNN Perturbation algorithm

Function Perturbation($listSol$, η)
Input Input solution $listSol$, strength of perturbation η
Output The perturbed solution $listSol$
$bestSol$, $otherSol$ ← getBestSolution($listSol$)
for each sol in $bestSol$ **do**
  for (**each** dimension i in sol) **do**
    if ($w_i >$ random and $i == 1$) **then**
      remove element i from sol
    end if
  end for
  sol ← RepairOperator(sol)
end for
for (each sol in $otherSol$) **do**
  for (i=1 to η) **do**
    Randomly remove an element from sol
  end for
  sol ← RepairOperator(sol)
end for
return $listSol$

4.5 Repair operator

When modifications are made to the solution through the K-means or KNN perturbation operators, the solution obtained must be verified with respect to compliance with its constraints and the possibility of adding new elements. To carry that out, a repair operator is used. As an input, the operator receives the solution to be repaired. The exit of the operator corresponds to the repaired solution. The first step in the procedure is to verify if the solution needs to be repaired. In the case that it needs to be repaired, by using the calculation defined by Eq. (19), we proceed to eliminate elements from the solution. Once the constraints are satisfied, the next step is to evaluate if new elements can be incorporated. For this purpose, we again use Eq. (19) to assign a weight to each element and identify the best element to incorporate. Once those procedures have been completed, the operator returns the repaired solution. The pseudo-code of this process is displayed in Algorithm 4.

Algorithm 4 Repair algorithm

Function Repair(sol)
Input Input solution sol
Output The Repair solution sol
$bneedRepair$ ← needRepair(sol)
while ($bneedRepair$ == True) **do**
  sol_{max} ← getMaxWeight(sol)
  sol ← removeElement(sol, sol_{max})
  $bneedRepair$ ← needRepair(sol)
end while
$state$ ← False
while ($state$ == False) **do**
  sol_{min} ← getMinWeight(sol)
  if ($sol_{min} == \emptyset$) **then**
    $state$ ← True
  else
    sol ← addElement(sol, sol_{min})
  end if
end while
return sol

5 Numerical results

In this section, the results of applying the KNN perturbation operator in the machine learning framework are provided. As a first step, we describe the methodology used to collect the parameters used in our algorithm, and then develop the experiments to evaluate its performance.

Our design and experimental development consist of two stages. The first stage corresponds to the identification and evaluation of the contribution of the KNN perturbation operator. The second stage aims to study how efficient our hybrid proposal is with respect to other binarization frameworks that have recently and efficiently solved the MKP.

The dataset[2] used to perform the experiments consists of instances that have 500 elements (n) and {5, 10, 30} constraints (m). The nomenclature used to identify an instance is: *mkp.m.n-x*, where m corresponds to the number of constraints, n to the total number of elements, and x to the instance. Each pair (m, n) gives rise to a dataset *mkp.m.n* where each one consists of 30 instances. In each *mkp.m.n* dataset, the 30 instances are divided into 3 groups depending on the constraints $b_i = t \times \sum_{j \in n} a_{ij}$,

[2] OR-Library: http://www.brunel.ac.uk/mastjjb/jeb/orlib/mknapinfo.html.

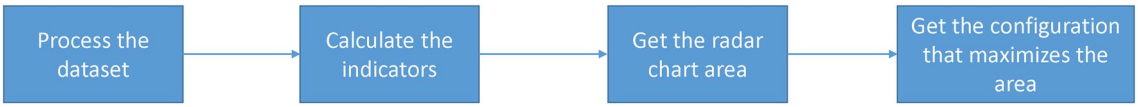

Fig. 4 Flow chart of parameter setting

Table 1 Parameter setting for the Cuckoo search algorithm

Parameters	Description	Value	Range
η	Perturbation operator coefficient	30%	[30, 40]
N	Number of nest	20	[20, 25]
G	Number of transition groups K-means operator	5	[4, 5, 6]
K	Neighbours number for the perturbation calculus	15	[10, 15]
γ	Step length	0.01	0.01
κ	Levy distribution parameter	1.5	1.5
Iteration number	Maximum iterations	800	[800]

where $t \in \{0.25, 0.50, 0.75\}$ corresponds to the tightness ratio. As algorithms to be binarized, cuckoo search and PSO were used. The choice is mainly due to the fact that they have been widely and successfully used to solve continuous optimization problems. For the execution of the instances, we use a computer equipped with an Intel Core i7-4770 processor with 16GB in RAM. The algorithm is programmed in Python 2.7.

5.1 Parameter settings

To perform the parametrization, we use 10 problem instances chosen randomly from dataset *mkp.5.250*. The range of parameters explored for the CS case is shown in Table 1, in the range column. The k used for the binarization operator and for the perturbation operator is included. The first step for obtaining the adequate parametrization is to process the data set with the machine learning swarm intelligence algorithm, for each parameter combination. Each combination executes the chosen instances of the dataset 10 times. With the averages of the 10 results obtained for each configuration, we calculate the 4 measurements defined in the Eqs. (22) to (25). Subsequently, for each combination, the four measurements are placed on a radar chart and the area of the chart is calculated. Finally, the configuration that obtains the largest area is chosen. The flow chart of the parameter settings is shown in Fig. 4.

1. The percentage deviation of the best value obtained in ten executions compared with the best known value:

$$bSolution = 1 - \frac{KnownBestValue - BestValue}{KnownBestValue} \quad (22)$$

2. The percentage deviation of the worst value obtained in ten executions compared with the best known value:

$$wSolution = 1 - \frac{KnownBestValue - WorstValue}{KnownBestValue} \quad (23)$$

3. The percentage deviation of the average value obtained in ten executions compared with the best known value:

$$aSolution = 1 - \frac{KnownBestValue - AverageValue}{KnownBestValue} \quad (24)$$

4. The convergence time for the best value in each experiment is normalized according to Eq. (25).

$$nTime = 1 - \frac{convergenceTime - minTime}{maxTime - minTime} \quad (25)$$

5.2 Contribution of the KNN perturbation operator

In this section, we describe the results of the experiments conducted to evaluate the contribution of the KNN perturbation operator. In the case of the K-means binarization, two additional operators were designed and built to have a baseline. These operators execute the binarization process in a random way taking into account different fixed probabilities of transition. A first operator considers a transition probability of 0.3 and a second operator a probability of 0.5. For the evaluation of the KNN perturbation operator, a random operator with a probability of perturbation given by the factor η was built additionally. We recall that the KNN perturbation operator, uses KNN along with the Morris measurement at the best 25 % of the population and the rest uses a random operator. In addition, we use a version of the algorithm, where the KNN perturbation operator is not considered. The nomenclature used is the following: *km.rand*.03 and *km.rand*.05 are the operators that

allow the evaluation of the binarization of the K-means operator using probability 0.3 and 0.5, respectively. A first operator considered a probability of transition of 0.3 and a second operator a probability of 0.5. The 0.3 value is used based on the result of a previous work García et al. [24]. This value is the one which has a better performance. In the case of 0.5, it is used because it is a random operator with the same probability of staying or making the transition. In the case of the KNN perturbation operator, we use the operator *knn.without* to represent the case in which we do not use the perturbation operator and *knn.random* in case we use the random perturbation operator. Our original algorithm is denoted with *knn.km*. In the case of the algorithms *km.rand*.03 and *km.rand*.05, we always use the KNN perturbation operator in its original form. In the case of the algorithms *knn.without* and *knn.random*, we always use the binarization operator K-means in its original form; these algorithms were studied in García et al. [24]. This way we analyze the incremental contribution of our approach.

For the execution of the experiments, the dataset *mkp.30.500* was used, which corresponds to the dataset of greatest complexity both in the number of restrictions and in the number of elements. Each instance of the dataset was executed 30 times to have statistical validity and the Wilcoxon signed-rank nonparametric test was used to evaluate if the difference between the results is significant. Table 2 shows the best value and the average value for all the variants mentioned above. In the Wilcoxon test, we compared *knn.km* which corresponds to our original algorithm with: *knn.without*, *knn.random*, *km.random*.03 and *km.random*.05. From Table 2 it is concluded that the operator *knn.km* is better in practically all instances for the Best value indicator. For the few cases where the variants have obtained a better value, to see if that value is robust, we use the average indicator. In those cases, our algorithm is superior in all instances, indicating that the KNN perturbation and K-means binarization operators, contribute in an important way to the robustness of our algorithm. The Wilcoxon test indicates that this robustness is statistically significant.

To have a better understanding of the contributions of KNN perturbation and km binarization operators to the final result, we use the indicators *Gap*(%) and *Bestvalue*(%) defined in the Eqs. (26) and (27), respectively. To simplify the visualization of the comparison, the *mkp.30.500* dataset was divided into three groups using the tightness ratio criterion described at the beginning of Section 5.

$$Gap(\%) = 100 \cdot \frac{KnownBestValue - Value}{KnownBestValue} \tag{26}$$

$$Bestvalue(\%) = 100 \cdot \frac{BestValue - Value}{BestValue} \tag{27}$$

Group 0 corresponds to problems contained between 0 and 9 including both and have a tightness ratio of 0.25. Group 1 contains the problems contained between 10 and 19 with a tightness ratio of 0.5. Finally, the instances contained between 20 and 29 correspond to group 2 with a tightness ratio of 0.75. In Fig. 5 the results are displayed using the *Gap*(%) indicator. With this indicator, we can identify in the three groups, how the KM operator contributes to improving the quality of the results. The *knn.random* and *knn.without* variants that use the KM operator to perform the binarization clearly have a better performance than *km.random.03* and *km.random.05* operators. The KNN perturbation operator contribution is also relevant. This is observed by comparing the variant *knn.km* with *knn.random* and *knn.without* in Fig. 5.

Additionally, to evaluate the satisfaction of the perturbation condition in the knn.km algorithm, a histogram was generated considering the times the criterion is executed. The result is shown in Fig. 6. From the histogram, it follows that for the largest number of instances the criterion is executed between 4 and 8 times. On the other hand, 2 was the minimum number of times the criterion was executed.

Because we consider different instances when making the violin charts, the *Gap*(%) indicator is not suitable to visualize the dispersion of the results. To evaluate this dispersion, we use the *Bestvalue*(%) indicator. This indicator considers the best value obtained by each of the variants and for each instance, to compare this with the value obtained from the variant in each of the executions. The results are shown in Fig. 7. In this figure, when we compare the interquartile ranges represented by dotted lines, we see that KNN perturbation and KM binarization operators contribute to decreasing the dispersion of the solutions in the three groups.

Our last experiment consists of evaluating the convergence of the different variants of our algorithm. For this, as in the previous cases, we separate the dataset into three groups. In each group, we consider the Best value obtained for each one of the executed instances every 80 iterations and we graph the %-Gap of this value. The results are shown in Fig. 8. In that figure, it can be observed that the convergence velocity is relatively similar for all the variants in the three groups. Therefore, there is no significant contribution from the KNN perturbation and KM binarization operators in the convergence of solutions.

6 Comparisons

This section aims to evaluate the performance of the *knn.km* algorithm against other binarization-based algorithms that have solved the MKP. As datasets, we use problems *mkp*.5.500 and *mkp*.10.500 of the OR-library. These instances after 30.500 correspond to the most difficult

Table 2 OR-Library benchmarks MKP *mkp*.30.500

Instance	Best	*knn.km*		*knn.random*		*knn.without*		*km.random*.03		*km.random*.05	
	Known	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg
0	116056	115868	115825.80	115526	115405.25	115524	115409.44	115128	114838.85	115280	115000.27
1	114810	114405	114372.65	114667	114540.88	114367	114285.68	114163	113962.07	114281	114087.12
2	116741	116583	116550.59	116158	116074.23	116142	116045.68	115976	115728.42	115970	115742.02
3	115354	115198	115172.37	114782	114707.85	114778	114700.28	114670	114435.14	114607	114353.86
4	116525	116353	116314.39	115995	115871.57	115939	115825.70	115794	115513.46	115794	115539.68
5	115741	115342	115295.06	115244	115130.98	115594	115501.41	114956	114718.22	115084	114845.46
6	114181	113987	113962.76	113593	113494.59	113624	113504.60	113298	113025.03	113315	113037.67
7	114348	114199	114177.87	113626	113498.46	113590	113467.74	113447	113176.09	113339	112990.34
8	115419	114822	114794.78	114800	114713.72	114822	114704.14	114667	114311.61	114613	114324.89
9	117116	116947	116903.70	116382	116296.70	116376	116264.31	116077	115692.70	116077	115796.44
10	218104	217995	217955.67	217629	217550.00	217776	217672.85	217530	217261.93	217556	217315.16
11	214648	214534	214498.96	214110	214023.88	213882	213767.02	213864	213624.73	213882	213495.42
12	215978	215638	215600.98	215588	215469.68	215690	215570.53	215534	215214.61	215534	215287.83
13	217910	217816	217768.78	217360	217283.73	217321	217193.52	217150	216945.18	217144	216772.01
14	215689	215152	215115.31	215119	214995.58	215119	215011.87	214992	214728.43	214916	214570.64
15	215919	215408	215360.90	215408	215314.51	215254	215167.29	215085	214782.77	215113	214850.59
16	215907	215576	215551.25	215453	215329.97	215516	215435.67	215394	215148.66	215314	214919.99
17	216542	216336	216313.94	216064	216000.58	215835	215719.01	215776	215580.09	215776	215426.45
18	217340	217013	216968.23	216816	216690.73	216962	216864.85	216872	216587.58	216882	216688.88
19	214739	214332	214288.23	214161	214087.18	214073	213999.96	214073	213858.33	214033	213658.82
20	301675	301343	301307.18	301347	301217.26	301296	301216.19	301240	301007.78	301219	301007.41
21	300055	299720	299682.27	299692	299566.19	299640	299521.07	299477	299272.86	299536	299261.83
22	305087	304852	304807.21	304815	304708.51	304815	304729.19	304639	304345.19	304663	304342.66
23	302032	301658	301635.93	301633	301550.11	301541	301434.63	301500	301307.75	301506	301315.42
24	304462	304186	304159.51	304149	304051.40	304173	304065.04	304060	303868.58	304048	303760.04
25	297012	296774	296738.24	296450	296387.08	296435	296318.37	296388	296028.17	296384	296068.89
26	303364	302941	302904.77	302899	302839.43	302833	302739.24	302666	302442.46	302666	302296.30
27	307007	306616	306581.56	306616	306487.11	306450	306326.48	306376	305987.95	306349	306047.10
28	303199	302791	302757.62	302572	302510.79	302572	302459.97	302506	302266.63	302470	302253.36
29	300596	300170	300122.15	300129	300055.02	300106	300030.51	300035	299709.86	299991	299604.25
Average	211451.87	211185.17	211149.62	210959.43	210861.77	210934.83	210831.74	210777.77	210512.37	210778.07	210488.69
p-value				1.86e-05	1.79e-05	1.73e-06	1.73e-06	6.34e-06	6.98e-06	1.73e-06	1.73e-06

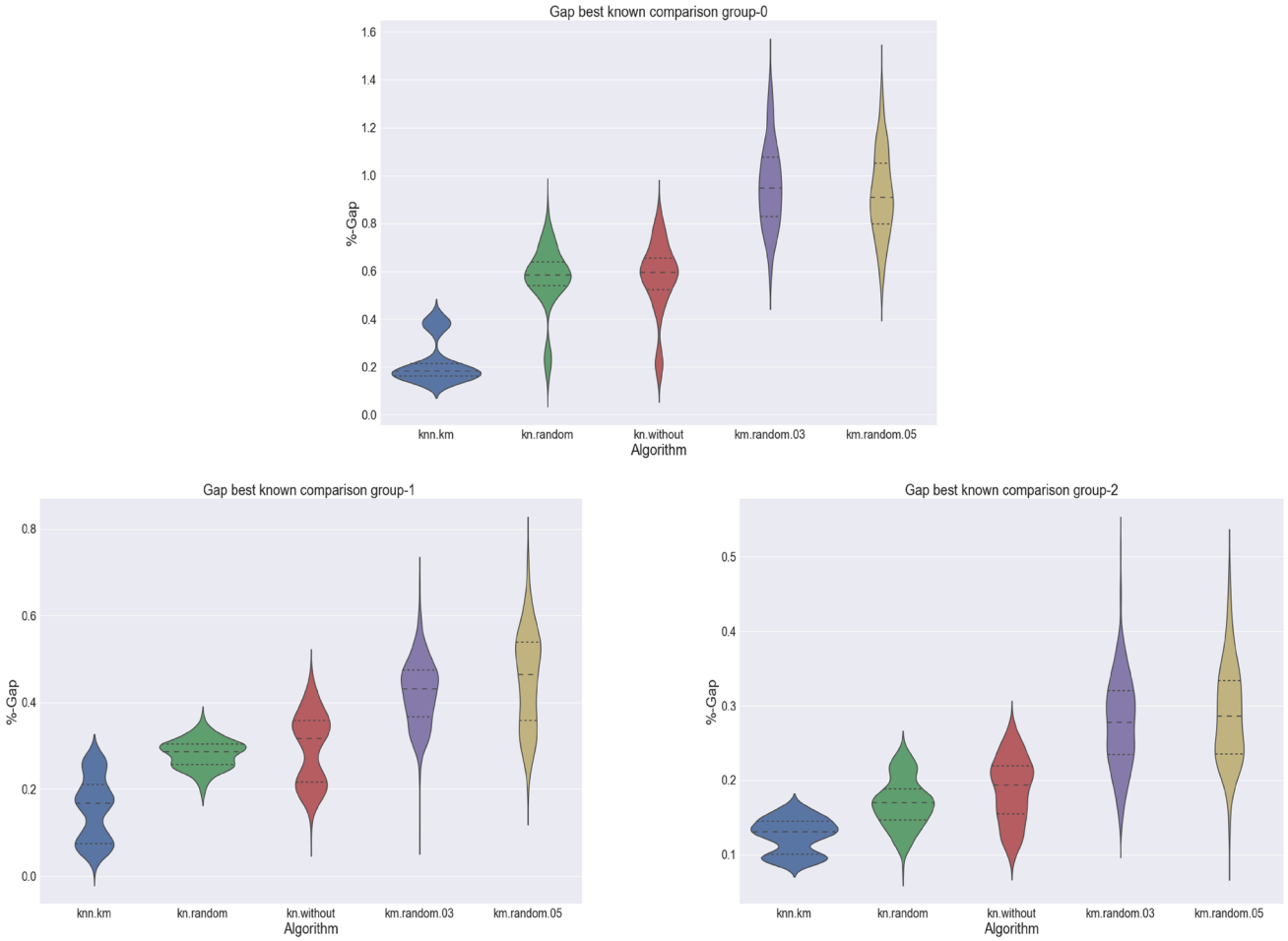

Fig. 5 %-Gap comparison between different algorithms for the *mkp.30.500* dataset

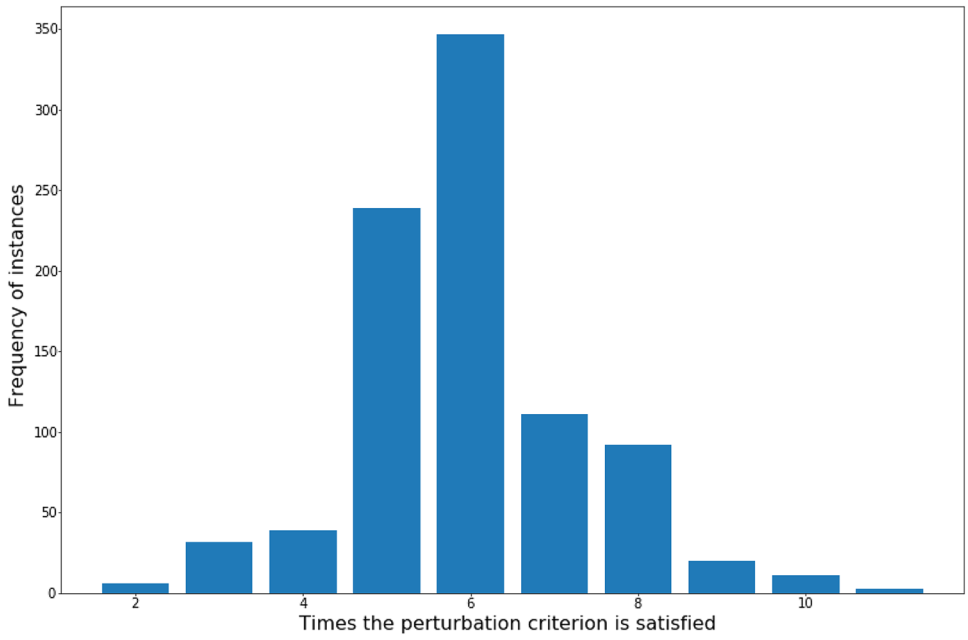

Fig. 6 Histogram with the number of perturbation operator executions for an instance

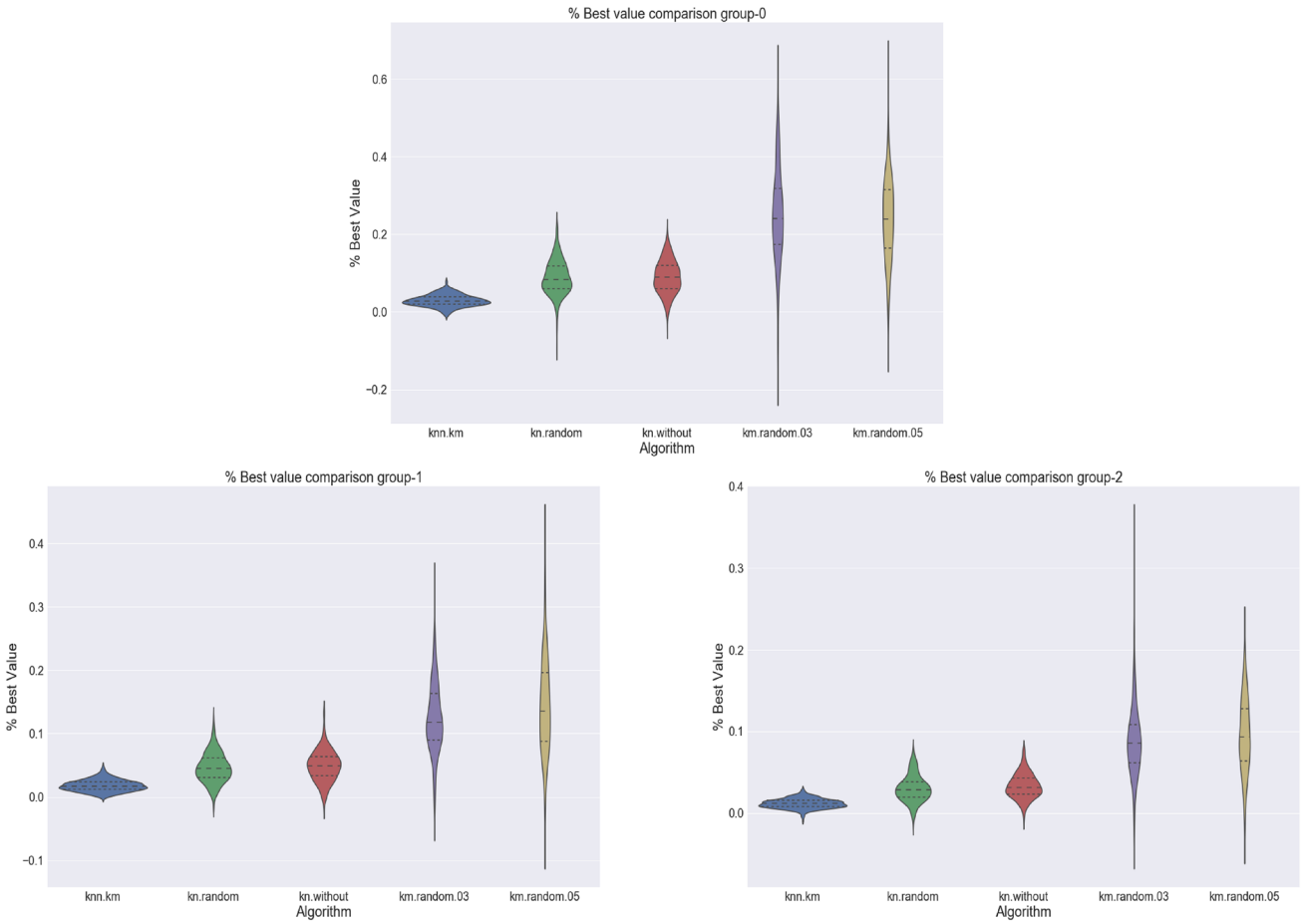

Fig. 7 %-Bestvalue comparison between different algorithms for the *mkp.30.500* dataset

instances of the library. For the evaluation, we choose according to the best of our knowledge the two best binarizations that use a general metaheuristics binarization mechanism based on transfer functions. The first of these algorithms corresponds to the Binary Artificial Algae Algorithm *BAAA* developed by Zhang et al. [61]. This algorithm uses a V-shape transfer function as a binarization mechanism. The second algorithm is a Binary differential search algorithm *BDS* developed by Liu et al. [37].

In Table 3, we evaluate the performance of our *knn.km* algorithm with *BAAA*.[3] It uses transfer functions as a general mechanism of binarization. In particular *BAAA* used the $\tanh = \frac{e^{\tau|x|}-1}{e^{\tau|x|}+1}$ function to perform the transfer. The parameter τ of the tanh function was set to a value 1.5. Additionally an elite local search procedure was used by *BAAA* to improve solutions. As maximum number of iterations *BAAA* used 35000. The computer configuration used to run the *BAAA* algorithm was: PC Intel Core(TM) 2 dual CPU Q9300@2.5GHz, 4GB RAM and 64-bit Windows 7 operating system. In our *knn.km* algorithm, the configurations are the same used in the previous experiments.

In addition, in order to determine if *knn.km* averages and standard deviations are significantly different than averages and deviations obtained by the *BAAA*, we have performed Student's *t*-test. The *t* statistic has the following form:

$$t = \frac{\hat{X_1} - \hat{X_2}}{\sqrt{\frac{(n_1-1)SD_1^2+(n_2-1)SD_2^2}{n_1+n_2-2}\frac{n_1+n_2}{n_1 n_2}}} \tag{28}$$

where:

$\hat{X_1}$: Average of *BAAA* for each instance

[3] Best values within our comparison are indicated in bold. This also holds for the next table.

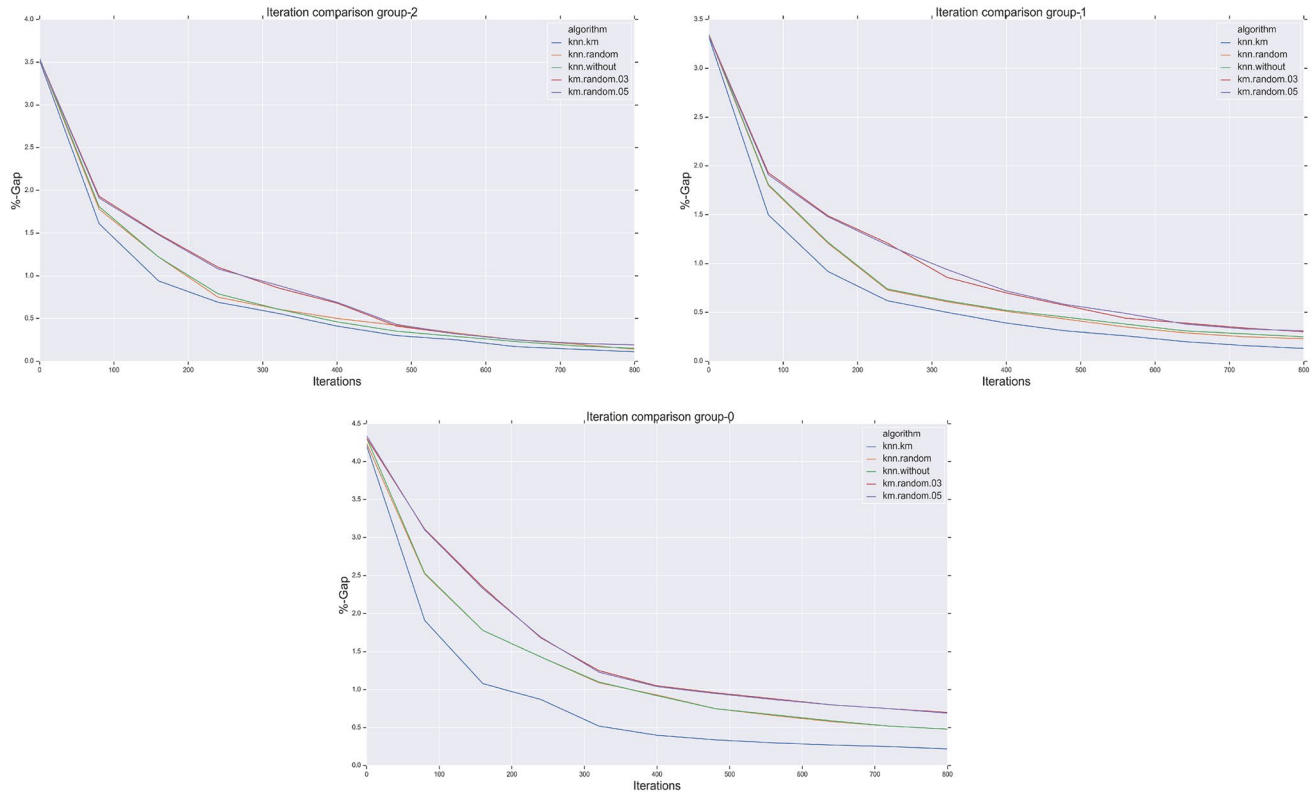

Fig. 8 Evolution of the best objective value for the different variants

SD_1: Standard deviation of *BAAA* for each instance
n_1: number of tests for *BAAA* for each instance
$\hat{X_2}$: Average of *knn.km-PSO* or *knn.km-CS* for each instance
SD_2: Standard deviation *knn.km-PSO* or *knn.km-CS* for each instance
n_2: number of tests for *knn.km-PSO* or *knn.km-CS* for each instance

The t values can be positive, neutral, or negative. The double-positive value (++) of t indicates that *knn.km* is significantly better than the other algorithms. In the opposite case (−−), *knn.km* obtains significantly worse solutions. If t is a single positive (+), *knn.km* shows to be better but not significantly. On the other hand, if the result is single negative (-), *knn.km* demonstrates to be worse, but not in a significant way. Finally, a neutral value of t depicts equality in the results. We stated confidence interval at the 95% confidence level. Finally, the Best value and the average results of all instances are compared with *knn.km* using the Wilcoxon test to evaluate if the results are significant in the whole dataset.

knn.km-PSO and *knn.km-CS* outperform practically in all problems the *BAAA*. In the Best value indicator, *knn.km-PSO* was higher in 13 instances, *knn.km-CS* in 8. Additionally in 8 instances *knn.km-PSO* and *knn.km-CS* obtained the same best value, and in one instance *BAAA* and *knn.km-PSO* obtained the best value. In the case of the average indicator, in 15 instances *knn.km-CS* outperform the other algorithms, in 13 instances it was *knn.km-PSO* and in one case it was *BAAA*.

In Table 4, we evaluate the performance of our *knn.km* algorithms with *TR-DBS* (tanh Random) and *TE-DBS* (tanh Elitist) developed in Liu et al. [37]. *DBS* used the $\tanh = \frac{e^{\tau|x|}-1}{e^{\tau|x|}+1}$ function to perform the binarization. The parameter τ of the tanh function was set to a value of 2.5. As maximum number of iterations *DBS* used 10000. For *DBS*, all computational experiments were conducted in Matlab 7.5 on a PC equipped with an Intel Pentium Dual-Core i7-4770 processor (3.40 GHz) with 16GB of RAM in the Windows OS. In our *knn.km*-framework, the configurations are the same used in the previous experiments.

The comparison between the *DBS* and *knn.km* algorithms show that for the Best value indicator, *TE-DBS* obtained 19 best values, *knn.km-CS* 8, *TR-DBS* 4 and *knn.km-PSO* 2. When the average indicator is observed, we see that *knn.Km-CS* scored 15 best average, *TE-DBS* scored 9, *knn.km-PSO* 5 and *TR-DBS* 1. The comparison

Table 3 OR-Library benchmarks MKP *mkp.5.500*

Instance	Best	*BAAA*			*knn.km-PSO*				*knn.km-CS*			
	Known	Best	Avg	std	Best	Avg	Time(s)	std	Best	Avg	Time(s)	std
0	120148	120066	120013.7	21.57	**120134**	**120089.3**	519	36.4++(9.8)	120096	120079.1	498	19.7++(12.3)
1	117879	117702	117560.5	11.4	**117844**	**117769.1**	541	41.4++(26.6)	117837	117758.1	527	46.2++(22.7)
2	121131	120951	120782.9	87.96	121039	120932.4	531	52.1++(8.0)	**121112**	**120961.3**	491	43.1++(8.4)
3	120804	120572	120340.6	106.01	**120752**	**120631.6**	499	63.2++(12.9)	**120752**	120644.1	467	73.2++(12.9)
4	122319	122231	122101.8	56.95	**122280**	122187.1	514	61.4++(5.6)	**122280**	**122201.4**	497	41.2++(7.8)
5	122024	121957	121741.8	84.33	**122007**	**121900.1**	568	46.1++(9.0)	121982	121841.2	497	31.3++(6.1)
6	119127	119070	118913.4	63.01	**119113**	118971.1	538	46.3++(4.0)	119094	**119001.1**	479	38.3++(6.5)
7	120568	120472	120331.2	69.09	120463	120341.1	549	55.1+(0.6)	**120536**	**120421.1**	481	51.1++(5.7)
8	121586	121052	120683.6	83.88	**121377**	121201.7	571	61.1++(27.3)	**121377**	**121231.2**	492	51.2++(30.5)
9	120717	120499	120296.3	110.06	120524	120401.3	591	73.2++(4.4)	**120685**	**120501.8**	472	48.5++(9.4)
10	218428	218185	217984.7	123.94	218296	218193.1	531	74.1++(7.9)	**218422**	**218281.5**	510	71.4++(11.3)
11	221202	220852	220527.5	169.16	**221007**	220918.1	598	68.9++(11.7)	**221007**	**220927.2**	548	64.1++(12.1)
12	217542	217258	217056.7	104.95	217356	217231.7	601	67.1++(9.3)	**217528**	**217427.1**	589	58.1++(16.9)
13	223560	223510	223450.9	26.02	**223558**	223471.7	621	41.2++(2.33)	223518	**223458.1**	531	42.1+(0.43)
14	218966	218811	218634.3	97.52	**218962**	218802.8	652	41.5++(8.7)	218884	**218807.3**	579	41.2++(9.0)
15	220530	220429	220375.9	31.86	**220514**	**220431.7**	669	47.2++(5.4)	220441	220361.3	541	28.7−(1.86)
16	219989	219785	219619.3	93.01	**219943**	219801.3	647	53.6++(9.3)	**219943**	**219802.1**	501	41.3++(9.8)
17	218215	218032	217813.2	115.37	218094	217891.3	693	55.1++(3.3)	**218194**	**217992.1**	603	51.2++(7.8)
18	216976	**216940**	**216862.0**	32.51	**216940**	216858.3	647	42.4−(0.4)	216873	216831.2	583	32.1−−(3.7)
19	219719	219602	219435.1	54.45	**219704**	**219621.6**	641	47.1++(14.1)	219693	219569.8	624	48.1++(10.1)
20	295828	295652	295505.0	76.30	**295717**	**295633.1**	631	42.1++(8.1)	**295717**	295631.3	541	34.3++(8.2)
21	308086	307783	307577.5	135.94	308065	307943.1	678	41.3++(14.1)	**308077**	**307957.1**	605	48.3++(14.5)
22	299796	299727	299664.1	28.81	**299796**	**299721.9**	711	64.1++(4.5)	299788	299681.3	537	46.7+(1,7)
23	306480	306469	306385.0	31.64	**306480**	**306448.3**	721	41.1++(6.7)	306476	306407.8	567	41.3++(2.4)
24	300342	300240	300136.7	51.84	**300245**	**300207.1**	712	31.1++(6.3)	**300245**	300197.8	649	31.5++(5.5)
25	302571	302492	302376.0	53.94	**302560**	**302471.8**	761	31.8++(8.4)	302492	302441.1	631	33.6++(5.6)
26	301339	301272	301158.0	44.3	**301322**	301251.7	769	35.4++(7.5)	**301322**	**301252.1**	649	35.8++(8.9)
27	306454	306290	306138.4	84.56	**306430**	**306326.1**	782	51.3++(10.4)	306422	306311.8	286	23.8++(10.8)
28	302828	302769	302690.1	34.11	**302822**	**302745.7**	757	31.6++(6.5)	302814	302727.1	614	31.3++(4.4)
29	299910	299757	299702.3	31.66	299828	299756.1	801	37.5++(6.0)	**299904**	**299786.7**	771	51.3++(7.7)
Average	214168.80	214014.23	213861.95	70.54	214105.73	214005.04	634.80	49.39	**214117.03**	**214016.41**	545.33	43.33
p-value with *knn.km-PSO*		3.16e-06	1.96e-06						0.76	0.65		
p-value with *knn.km-CS*		1.08e-05	3.52e-06		0.76	0.65						

Table 4 OR-Library benchmarks MKP *mkp.10.500*

Instance	Best	*TR-DBS*		*TE-DBS*		*knn.km-PSO*			*knn.km-CS*		
	Known	Best	Avg	Best	Avg	Best	Avg	Time(s)	Best	Avg	Time
0	117821	114716	114425.4	**117811**	**117801.2**	117779	117683.5	763	117801	117697.2	789
1	119249	119232	**119223.0**	**119249**	118024.0	119232	119048.2	828	119200	118988.1	761
2	119215	119215	117625.6	**119215**	117801.4	119194	**118876.4**	819	119159	118849.7	794
3	118829	**118813**	117625.8	**118813**	117801.2	**118813**	**118731.8**	783	118802	118701.3	789
4	116530	114687	114312.4	**116509**	114357.2	116434	116168.9	774	116471	**116218.3**	812
5	119504	119504	112503.7	**119504**	117612.8	119483	**119301.2**	811	119442	119297.2	805
6	119827	116094	115629.1	**119827**	**119827.4**	119749	119602.4	769	119764	119608.1	817
7	118344	116642	115531.9	118301	117653.3	118307	118141.8	825	**118309**	**118145.6**	808
8	117815	114654	114204.0	**117815**	115236.4	117801	117577.3	775	117781	**117588.1**	794
9	119251	114016	113622.8	**119231**	118295.1	119186	**118961.8**	826	119196	118951.2	817
10	217377	209191	208710.2	**217377**	212570.3	217318	217065.1	842	217343	**217064.6**	837
11	219077	**219077**	217277.2	**219077**	218570.2	219036	218901.7	848	219022	**218967.7**	837
12	217847	210282	210172.3	217377	212570.4	217772	217599.2	859	**217797**	**217691.4**	884
13	216868	209242	206178.6	**216868**	**216868.9**	216843	216603.2	892	216802	216651.3	901
14	213873	207017	206656.0	207017	206455.0	**213814**	**213524.1**	923	213809	213511.2	912
15	215086	204643	203989.5	**215086**	**215086.0**	215013	214811.3	821	215021	214931.3	887
16	217940	205439	204828.9	**217940**	**217940.5**	217825	217699.1	924	217880	217674.8	912
17	219990	208712	207881.6	**219984**	209990.2	219825	219547.3	922	219949	**219601.3**	911
18	214382	210503	209787.6	210735	211038.2	214332	213989.1	886	**214346**	**214014.8**	896
19	220899	205020	204435.7	**220899**	219986.8	220833	220572.1	967	220827	**220588.3**	1002
20	304387	**304387**	302658.8	304387	**304264.5**	304344	304012.7	1007	304351	304062.6	973
21	302379	**302379**	301658.6	302379	**302164.4**	302332	302101.6	1004	302263	302177.8	996
22	302417	290931	290859.9	**302416**	302014.6	302354	302081.7	982	302354	**302121.5**	995
23	300784	290859	290021.4	291295	291170.6	300743	300497.1	1038	**300745**	**300546.6**	1066
24	304374	289365	288950.1	**304374**	**304374.0**	304267	304173.1	858	304340	304194.7	984
25	301836	292411	292061.8	**301836**	**301836.0**	301730	301604.5	995	301754	301610.4	1084
26	304952	291446	290516.2	291446	291446.0	304905	304783.8	1081	**304911**	**304817.1**	1012
27	296478	293662	293125.5	295342	294125.5	296361	296201.3	1047	**296437**	**296307.2**	1028
28	301359	285907	285293.4	288907	287923.4	301293	301073.2	988	**301313**	**301112.6**	1074
29	307089	290300	289552.4	295358	290525.2	307002	306837.2	1102	**307014**	**306901.3**	974
Average	212859.3	206278.2	205310.6	210879.2	209511.0	212797.3	212592.4	898.6	**212806.8**	**212619.8**	905.0
p-value with *knn.km-PSO*		1.86e-05	1.9e-06	0.79	2.2e-04				0.19	2.1e-03	
p-value with *knn.km-CS*		2.59e-05	1.9e-06	0.69	1.8e-04	0.19	2.1e-03				

between *DBS* and the *knn.km* algorithms shows that for the Best value indicator *TE-DBS* obtained 19 best values, *knn.km-CS* 8, *TR-DBS* 4 and *knn.km-PSO* 2. The average indicator also indicates that *knn.km-CS* obtained 15 best average, *TE-DBS* obtained 9, *knn.km-PSO* 5 and *TR-DBS* 1. Furthermore, the Wilcoxon test in the case of the Best value indicator points out significant differences only between *knn.km-PSO* and *TR-DBS* and between *knn.km-CS* and *TR-DBS*, both cases in favor of the *knn.km* algorithms. When we evaluate the average indicator, we observe that *knn.km-CS* has a significant difference over all other algorithms and, in the case of *knn.km-PSO*, it exhibits significant differences over *TR-DBS* and *TE-DBS*.

7 Conclusions

In this work, we have proposed an improved binarization framework, which uses the K-means technique to enable continuous metaheuristics for COPs. The proposed framework integrates a local perturbation operator based on the K-nearest neighbor technique. Using of this approach, particle swarm optimization and cuckoo search metaheuristics were used to solve the well-known multidimensional knapsack problem. In this regard, the computational results in the 90 largest instances commonly used in the literature showed that the proposed improved framework

works competitively compared with the best algorithms that implement general binarization techniques. The KNN perturbation operator of the algorithm was analyzed and it was shown that the operator played a key role in the performance of the algorithm. In particular, for the case of the largest dataset *mkp*.30.500, the use of this operator allows improving the results going from average values of 0.14 % when a random perturbation operator is used to values of 0.28 % when the operator is replaced with a KNN perturbation operator. Finally, when comparing our algorithmic approach with the best binarization frameworks proposed so far for solving the multidimensional knapsack problem, it could be observed that our approach is able to improve their previous results.

As future work, we plan to investigate the behavior of other continuous metaheuristics when introduced in our framework. Additionally, it is interesting to apply and assess the performance of these hybrid algorithms to other $\mathcal{NP}$ -hard combinatorial problems. Another interesting topic worth investigation would be a comparison of different automatic parametrization methods including the one used in this paper and, e.g, IRACE López-Ibáñez et al. [39]. In addition, it seems interesting to investigate the use of machine learning techniques to operators that control the size of the population in order to reduce convergence times of the optimization algorithms. These techniques could find areas where it is convenient to intensify or diversify and areas where it is not appropriate to do so. On the other hand, we find it challenging to extend the current work designing perturbation operators which use artificial neural network techniques. In principle, we can explore perturbation operators which use traditional neural networks such as the feedforward or recursive neural networks and later extend the perturbation operators to use neural networks with reinforcement learning. Finally, it seems useful to explore hybridization in another direction, using metaheuristic algorithms that allow finding the proper parameterization and topology of a neural network automatically when it tries to solve a specific data problem.

Acknowledgements José García was supported by the Grant CONICYT/FONDECYT/INICIACION/11180056.

References

1. Ahmad SR, Bakar AA, Yaakub MR (2015) Metaheuristic algorithms for feature selection in sentiment analysis. In: Science and Information Conference (SAI), pp 222–226. IEEE
2. Al-Madi N, Faris H, Mirjalili S (2019) Binary multi-verse optimization algorithm for global optimization and discrete problems. Int J Mach Learn Cybern 1–21
3. Asta S, Özcan E, Curtois T (2016) A tensor based hyper-heuristic for nurse rostering. Knowl-Based Syst 98:185–199
4. Cadenas JM, Garrido MC, Muñoz E (2009) Using machine learning in a cooperative hybrid parallel strategy of metaheuristics. Inf Sci 179(19):3255–3267
5. Calvet L, de Armas J, Masip D, Juan AA (2017) Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. Open Math 15(1):261–280
6. Caserta M, Voß S (2010) Metaheuristics: intelligent problem solving. Springer, Boston, pp 1–38
7. Chen E, Li J, Liu X (2011) In search of the essential binary discrete particle swarm. Appl Soft Comput 11(3):3260–3269
8. Chou J-S, Ngo N-T, Pham A-D (2015) Shear strength prediction in reinforced concrete deep beams using nature-inspired metaheuristic support vector regression. J Comput Civ Eng 30(1):04015002
9. Chou J-S, Nguyen T-K (2018) Forward forecast of stock price using sliding-window metaheuristic-optimized machine-learning regression. IEEE Trans Industr Inf 14(7):3132–3142
10. Chou J-S, Pham A-D (2017) Nature-inspired metaheuristic optimization in least squares support vector regression for obtaining bridge scour information. Inf Sci 399:64–80
11. Chou J-S, Thedja JPP (2016) Metaheuristic optimization within machine learning-based classification system for early warnings related to geotechnical problems. Autom Constr 68:65–80
12. Crawford B, Soto R, Astorga G, García J, Castro C, Paredes F (2017) Putting continuous metaheuristics to work in binary search spaces. Complexity, 2017:Article ID 8404231
13. Damaševičius R, Woźniak M (2017) State flipping based hyper-heuristic for hybridization of nature inspired algorithms. In: International Conference on Artificial Intelligence and Soft Computing, pp 337–346. Springer
14. de Alvarenga Rosa R, Machado AM, Ribeiro GM, Mauri GR (2016) A mathematical model and a clustering search metaheuristic for planning the helicopter transportation of employees to the production platforms of oil and gas. Comput Ind Eng 101:303–312
15. De Jong K (2007) Parameter setting in EAs: a 30 year perspective. Parameter setting in evolutionary algorithms. Springer, Berlin, pp 1–18
16. de León AD, Lalla-Ruiz E, Melián-Batista B, Moreno-Vega JM (2017) A machine learning-based system for berth scheduling at bulk terminals. Expert Syst Appl 87:170–182
17. Fernandes S, Setoue K, Adeli H, Papa J (2017) Fine-tuning enhanced probabilistic neural networks using metaheuristic-driven optimization. In: Bio-Inspired Computation and Applications in Image Processing, pp 25–45. Elsevier
18. Fong S, Wong R, Vasilakos AV (2016) Accelerated PSO swarm search feature selection for data stream mining big data. IEEE Trans Serv Comput 9(1):33–45
19. Fréville A (2004) The multidimensional 0–1 knapsack problem: an overview. Eur J Oper Res 155(1):1–21
20. García J, Altimiras F, Peña A, Astorga G, Peredo O (2018a) A binary cuckoo search big data algorithm applied to large-scale crew scheduling problems. Complexity, 2018:Article ID 8395193
21. García, J, Crawford B, Soto R, Astorga G (2017) A percentile transition ranking algorithm applied to knapsack problem. In: Proceedings of the Computational Methods in Systems and Software, pp 126–138. Springer
22. García J, Crawford B, Soto R, Astorga G (2018b) A percentile transition ranking algorithm applied to binarization of continuous swarm intelligence metaheuristics. In: International Conference on Soft Computing and Data Mining, pp 3–13. Springer
23. García J, Crawford B, Soto R, Astorga G (2019a) A clustering algorithm applied to the binarization of swarm intelligence continuous metaheuristics. Swarm Evol Comput 44:646–664
24. García J, Crawford B, Soto R, Castro C, Paredes F (2018c) A k-means binarization framework applied to multidimensional knapsack problem. Appl Intell 48(2):357–380

25. García J, Moraga P, Valenzuela M, Crawford B, Soto R, Pinto H, Peña A, Altimiras F, Astorga G (2019b) A db-scan binarization algorithm applied to matrix covering problems. Comput Intell Neurosci, 2019
26. Garey M, Johnson D (1979) A guide to the theory of NP-completeness. Comput Intractability
27. Guo H, Liu B, Cai D, Lu T (2018) Predicting protein-protein interaction sites using modified support vector machine. Int J Mach Learn Cybernet 9(3):393–398
28. Iooss B, Lemaître P (2015) A review on global sensitivity analysis methods. Uncertainty management in simulation-optimization of complex systems. Springer, Berlin, pp 101–122
29. Jin Y, Qu R, Atkin J (2014) A population-based incremental learning method for constrained portfolio optimisation. In: Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2014 16th International Symposium on, pp 212–219. IEEE
30. Juan AA, Faulin J, Grasman SE, Rabe M, Figueira G (2015) A review of simheuristics: extending metaheuristics to deal with stochastic combinatorial optimization problems. Oper Res Perspect 2:62–72
31. Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In: Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on, vol 5, pp 4104–4108. IEEE
32. Korkmaz S, Babalik A, Kiran MS (2018) An artificial algae algorithm for solving binary optimization problems. Int J Mach Learn Cybernet 9(7):1233–1247
33. Kuo R, Kuo P, Chen YR, Zulvia FE (2016a) Application of metaheuristics-based clustering algorithm to item assignment in a synchronized zone order picking system. Appl Soft Comput 46:143–150
34. Kuo R-J, Mei C, Zulvia FE, Tsai C (2016b) An application of a metaheuristic algorithm-based clustering ensemble method to APP customer segmentation. Neurocomputing 205:116–129
35. Leonard BJ, Engelbrecht AP, Cleghorn CW (2015) Critical considerations on angle modulated particle swarm optimisers. Swarm Intell 9(4):291–314
36. Li Z-q, Zhang H-l, Zheng J-h, Dong M-j, Xie Y-f, Tian Z-j (2011) Heuristic evolutionary approach for weighted circles layout. Information and automation. Springer, Berlin, pp 324–331
37. Liu J, Wu C, Cao J, Wang X, Teo KL (2016) A binary differential search algorithm for the 0–1 multidimensional knapsack problem. Appl Math Model 40(23–24):9788–9805
38. Liu W, Liu L, Cartes D (2007) Angle modulated particle swarm optimization based defensive islanding of large scale power systems. IEEE Power Engineering Society Conference and Exposition in Africa 1–8
39. López-Ibáñez M, Dubois-Lacoste J, Cáceres LP, Birattari M, Stützle T (2016) The irace package: Iterated racing for automatic algorithm configuration. Oper Res Perspect 3:43–58
40. Mann PS, Singh S (2017) Energy efficient clustering protocol based on improved metaheuristic in wireless sensor networks. J Netw Comput Appl 83:40–52
41. Martin S, Ouelhadj D, Beullens P, Ozcan E, Juan AA, Burke EK (2016) A multi-agent based cooperative approach to scheduling and routing. Eur J Oper Res 254(1):169–178
42. Mirghasemi S, Andreae P, Zhang M (2019) Domain-independent severely noisy image segmentation via adaptive wavelet shrinkage using particle swarm optimization and fuzzy c-means. Expert Syst Appl 133:126–150
43. Moiz DZ, AE, Mezioud C, Draa A (2015) Binary bat algorithm: On the efficiency of mapping functions when handling binary problems using continuous-variable-based metaheuristics. In: Computer Science and Its Applications - 5th IFIP TC 5 International Conference, CIIA 2015, Saida, Algeria, May 20-21, 2015, Proceedings, pp 3–14
44. Pampara G (2012) Angle modulated population based algorithms to solve binary problems. PhD thesis, University of Pretoria, Pretoria
45. Ries J, Beullens P (2015) A semi-automated design of instance-based fuzzy parameter tuning for metaheuristics based on decision tree induction. J Oper Res Soc 66(5):782–793
46. Santos HG, Ochi LS, Marinho EH, Drummond LMDA (2006) Combining an evolutionary algorithm with data mining to solve a single-vehicle routing problem. Neurocomputing 70(1–3):70–77
47. Saremi S, Mirjalili S, Lewis A (2015) How important is a transfer function in discrete heuristic algorithms. Neural Comput Appl 26(3):625–640
48. Smith-Miles K, Baatar D, Wreford B, Lewis R (2014) Towards objective measures of algorithm performance across instance space. Comput Oper Res 45:12–24
49. Streichert F, Stein G, Ulmer H, Zell A (2003) A clustering based niching method for evolutionary algorithms. Genetic and evolutionary computation conference. Springer, Berlin, pp 644–645
50. Swagatam D, Rohan M, Rupam K (2013) Multi-user detection in multi-carrier cdma wireless broadband system using a binary adaptive differential evolution algorithm. Proceedings of the 15th annual conference on Genetic and evolutionary computation, GECCO, pp 1245–1252
51. Talbi E-G (2016) Combining metaheuristics with mathematical programming, constraint programming and machine learning. Ann Oper Res 240(1):171–215
52. Tsai C-W, Chiang M-C, Ksentini A, Chen M (2016) Metaheuristic algorithms for healthcare: open issues and challenges. Compu Electr Eng 53:421–434
53. Tyasnurita R, Özcan E, Shahriar A, John R (2015) Improving performance of a hyper-heuristic using a multilayer perceptron for vehicle routing. In: 15th UK Workshop on Computational Intelligence, UK
54. Vasquez M, Hao J-K (2001) A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. Comput Optim Appl 20(2):137–157
55. Veček N, Mernik M, Filipič B, Črepinšek M (2016) Parameter tuning with chess rating system (CRS-tuning) for meta-heuristic algorithms. Inf Sci 372:446–469
56. Xue Y, Xue B, Zhang M (2019) Self-adaptive particle swarm optimization for large-scale feature selection in classification. ACM Trans Knowl Discov Data (TKDD) 13(5):50
57. Yalcinoz T, Altun H (2001) Power economic dispatch using a hybrid genetic algorithm. IEEE Power Eng Rev 21(3):59–60
58. Yang M-H (2001) An efficient algorithm to allocate shelf space. Eur J Oper Res 131(1):107–118
59. Yang Y, Mao Y, Yang P, Jiang Y (2013) The unit commitment problem based on an improved firefly and particle swarm optimization hybrid algorithm. In: Chinese Automation Congress (CAC), 2013, pp 718–722. IEEE
60. Zhang G (2011) Quantum-inspired evolutionary algorithms: a survey and empirical study. J Heuristics 17(3):303–351
61. Zhang X, Wu C, Li J, Wang X, Yang Z, Lee J-M, Jung K-H (2016) Binary artificial algae algorithm for multidimensional knapsack problems. Appl Soft Comput 43:583–595
62. Zheng B, Zhang J, Yoon SW, Lam SS, Khasawneh M, Poranki S (2015) Predictive modeling of hospital readmissions using metaheuristics and data mining. Expert Syst Appl 42(20):7110–7120