Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

A KNN quantum cuckoo search algorithm applied to the multidimensional knapsack problem

José García [*], Carlos Maureira

Escuela de Ingeniería en Construcción, Pontificia Universidad Católica de Valparaíso, 2362807 Valparaíso, Chile

A R T I C L E   I N F O

Article history:
Received 5 August 2020
Received in revised form 26 December 2020
Accepted 29 December 2020
Available online 4 January 2021

Keywords:
Combinatorial optimization
Machine learning
Metaheuristics
KNN
Multidimensional knapsack problem

A B S T R A C T

Optimization algorithms and particularly metaheuristics are constantly improved with the goal of reducing execution times, increasing the quality of solutions, and addressing larger target instances. Hybridizing techniques are one of these methods particularly interesting for the broad scope of problems to which they can be adapted. In this work, we assessed a hybrid algorithm that uses the k-nearest neighbor technique to improve the results of a quantum cuckoo search algorithm for resource allocation. The k-nearest neighbor technique is used to direct the movement of solutions. Numerical experiments were performed to obtain insights from the contribution of the k-nearest neighbor technique in the final result of solutions. The well-known multidimensional knapsack problem was addressed in order to validate our procedure; a comparison is made with state-of-the-art algorithms. Our results show that our hybrid algorithm consistently produces better results in most of the analyzed instances.

1. Introduction

In recent years, metaheuristics have shown their efficiency in tackling complex problems, including complex combinatorial problems. Examples can be found in logistics [1], civil engineering [2], biology [3], machine learning [4], among others. Despite the improved efficiencies and due in part to the large size of many combinatorial problems, it is also necessary for the continuous strengthening of metaheuristics methods. Thus, hybrid methods have been used for improving metaheuristics algorithms.

Among the main hybridization techniques, we find *hybrid heuristics*, [5], where different metaheuristics algorithms are combined to enhance their capabilities. For example, to solve the ordering planning problem in [6], the authors used a simulated annealing-based genetic and tabu-search-based genetic algorithms. In this study, the hybrid approaches were compared with the traditional approaches, showing the first ones a better performance than the traditional approaches.

Matheuristics, [7], where mathematical programming methods are integrated along metaheuristic techniques. For example, in [8], the vehicle routing problem was studied using mixed-integer linear programming and matheuristic approaches. Particularly in the matheuristic approach, it was used with the aim of applying it to large problems. In this approach, the original problem was divided into hierarchical subproblems: assignment problem, sequence problem, and scheduling problem.

Simheuristics, [9], which embraces a combination of simulations with metaheuristics modeling. In [10], the project portfolio selection problem was addressed incorporating an uncertain condition. The simulation–optimization algorithm was used to model uncertainty. This algorithm integrates the Monte Carlo simulation with variable neighborhood search metaheuristic.

Notably, these methods rarely exploit the ancillary data generated by metaheuristics to get more robust results. Metaheuristics generate important accessory data in the solutions-search process, and this data can be exploited by informing machine learning methods. Integrating machine learning techniques with metaheuristic algorithms is a new line of research that has become important in recent years. A detailed review of the integration between machine learning and metaheuristics will be developed in Section 2.

Aligned with these lines of interaction between machine learning and metaheuristics, we proposed a hybrid algorithm that uses the k-nearest neighbor technique embedded in the motion of a quantum cuckoo search algorithm. The proposed algorithm integrates two original tactics, which allow using the data generated in the execution of the MH to get better results. This hybridization aims to improve the quality of the solutions obtained when applying the algorithm to the multidimensional knapsack problem. The contribution of this work includes:

1. A KNN Quantum-Cuckoo-Search-Algorithm (KQCSA), integrated with the k-nearest neighbor (KNN) technique is

[*] Corresponding author.
E-mail addresses: jose.garcia@pucv.cl (J. García), carlos.maureira@gmail.com (C. Maureira).

https://doi.org/10.1016/j.asoc.2020.107077
1568-4946/© 2021 Elsevier B.V. All rights reserved.

proposed. Cuckoo search (CS) was used as an optimization algorithm, however, the method can easily be adapted to other algorithms.
2. KQCSA is further strengthened with a solution update mechanism which also uses the KNN technique.
3. For a proper evaluation of the contribution of the KNN proposed operator, numerical experiments were carried out allowing us to statistically assess their contribution to the method.
4. Our KQCSA is applied to the well-known multidimensional knapsack problem. In particular, we tested our method against large data instances and compared them with different state-of-the-art methods to evaluate its efficiency. For this purpose, we use the largest problems of the OR-Library. The numerical results show that our hybrid algorithm achieves highly competitive results.

A brief structure of contents is: In Section 2, we briefly overview the state-of-the-art in integrating metaheuristics algorithms and machine learning techniques. The knapsack problem is detailed in Section 3. In Section 4, the detail of the KNN quantum cuckoo search algorithm is developed. Our numerical experiments and comparisons are detailed in Section 5. Finally, in Section 6, the conclusions and future lines of research are discussed.

2. Related work

This section aims to develop a brief state-of-the-art of integration between machine learning algorithms and metaheuristics techniques. For a proper understanding, three categories are identified: Low-level integrations, high-level integrations, and optimization problems. Additionally, Table 1 contains a summary of the selected articles.

- Metaheuristics advantages and applications:
  Due to good performance in complex problems, particularly where exact techniques do not perform well, the area of metaheuristic (MH) algorithms has grown significantly in the last decade. Applications of MH techniques have been developed in areas such as sustainability [11], finance [12], logistics [13], civil engineering [14], feature selection [15], among others.
- Opportunities for Machine Learning into Metaheuristics:
  The application of metaheuristic algorithms includes, in the process of searching for the best solutions, the generation of a significant amount of ancillary data. Notably, in the iterative search mechanism, data on trajectories, solutions, fitness, and movements in the solution space are generated. The access to these data sets open opportunities for machine learning (ML) techniques to be applied to improve the quality and the convergence times of the solutions obtained by metaheuristic algorithms.
- ML enhancing MH:
  There are three main categories in which ML algorithms use the data generated by metaheuristics [16,17], see Table 1 for examples of Categories of ML enhancing MH algorithms.

  – **Low level:**. The low-level integrations in which ML techniques improve specific operators of the MH algorithm, examples include: local search operators, population initiation, binarization, parameter tuning, etc.

    ∗ Parameter tuning: An iterated racing procedure was used in [18]; a reset mechanism in combination with an elitist procedure (to ensure the best configuration) and the use of truncated sampling distribution allowed the automatic configuration of parameters. Similarly, decision trees have been also used to tune specific parameters for the case of the traveling salesman problem [19]. The decision tree method managed to improve the quality of the solutions and the computational time when compared with fixed parameter settings.
    ∗ Initialization of solutions: Decision trees were used to generate initial solutions, for new instances, in junction with Opposition Based Learning (OBL), to initiate complementary solutions, for the shop scheduling problem [20]. Other techniques such as Reinforcement Learning (RL) also can be used to build solutions. In these cases, the solution construction can be seen as a successive addition of decisions, where a RL algorithm can be trained in the construction of these. Particularly in [21], a Deep Q-network was built and applied to the optimization of the Job-Shop Scheduling Problem. Similarly, transfer learning techniques in [22], were used for initial solutions with three evolutionary multi-objective algorithms and applied to 12 benchmark functions.
    ∗ Binarization. Another active area of research aims to construct binary versions of algorithms that work naturally in continuous spaces. In this area, there are examples of integration between ML and metaheuristics. In [23] the K-means technique was used to generate binary versions of the cuckoo search algorithm and applied to the matrix covering problem. Whereas in the civil engineering area, in [14], a hybrid algorithm designed using db-scan as a binarization method and applied to optimize the emission of CO_2 on retaining walls.

  – **High level**. In this category, ML techniques do work by selecting different metaheuristic algorithms, i.e. choosing the most appropriate one for each instance. High-level interactions mostly allow machine learning techniques to identify the best optimization algorithm, solving particular instances in the case of having sets of optimization algorithms. Some examples are:

    ∗ Cooperative strategies: The aim is to combine different algorithms in parallel or sequential form. When these combinations are performed properly, cooperative strategies allow to obtain more robust optimization algorithms.
      In [24], a cooperative parallel metaheuristics strategy is proposed with the intention of efficiently addressing the high school timetabling problem. The algorithm simultaneously exploits the generation of time-columns and a set of metaheuristics in order to obtain better solutions. This cooperative strategy had a better performance than state-of-the-art methods.
    ∗ Algorithm selection: In the case of algorithm selection, the goal is to choose from a portfolio of algorithms along with a set of characteristics associated with each problem instance.
      For the Bulk Berth Allocation Problem, the quality of the solutions is increased considerably [25]. Whereas in the case of the vehicle routing problem with time windows, the inclusion of ML outperforms previous results [26]. A parallel multi-objective algorithm was used to find high-quality

settings autonomously, surpassing state-of-the-art algorithms, when applied to Vehicle Network Communications [27]. A multi-agent cooperative framework was proposed in [28] to address routing and scheduling problems. In this framework, each agent has the possibility to instantiate a different metaheuristic and then agents can partially share the solutions. The algorithm produced new best-known solutions in the case of the capacitated vehicle routing problem.

  * Hyper-heuristic: In the case of hyper-heuristic, the objective is to automate the design and selection of a heuristic or metaheuristic in order to apply it to a wider spectrum of problems. For the vehicle routing problem, a multilayer perceptron neural network was designed to improve the state-of-the-art selection hyper-heuristics performance [29]. Q-learning Reinforcement learning was used in [30] in order to guide the hyper-heuristic model in the selection of the appropriate components in the different stages of optimization. Whereas in [31], a learning greedy algorithm was applied to a hybrid, flexible flow shop problem; showing significant improvements compared with different state-of-the-art methods.

- **Optimization problem**. In this case, ML techniques help to model the objective function or the constraints of the optimization problem. The latter allows for reformulating the optimization problem, incorporating, for example, a measure of the quality of the intermediate solutions. They can be applied also to the problem's specific constraints, or on the selection of relevant characteristics to the problem at hand. The first group, reformulating the optimization problem, relates to the prediction of the objective function; for example, when the objective function has a high computational cost in its evaluation.

  * Objective function: Optimizing cost and CO_2 emissions, in designing a concrete box-girder bridge, the kriging method modeled the objective function. This allows reducing the computation times and improving the solutions obtained by traditional methods [32]. In [33], neural networks were used to model viscosity and conductivity values and further integrated into NSGA-II (nondominated sorting genetic algorithm II) to perform optimization. Similarly, neural networks were applied to predict the energy consumption for heating, ventilation, and air conditioning systems in buildings. Then, a multi-objective genetic algorithm was used to find the optimal consumption conditions [34]. As a result, the multi-objective optimization shows better results in terms of thermal comfort and energy consumption when compared to the base case design.
  * Constraints: Neural networks have also been used to transform constrained optimization problems into unconstrained problems. For example, in [35], the economic optimization problem for the initial rainwater storage tank is transformed into an unconstrained optimization problem with a neural network in conjunction with particle swarm optimization (PSO). The results show improvements in the costs of water treatment when compared with the standard methods. Industrial optimization problems are also aid when solutions do not meet the restrictions and there is no information as to why. For example, in [36], the authors use a decision tree in order to identify and repair solutions that do not satisfy the constraints. The proposed methods outperformed the baseline algorithm, which does not use repair methods.

Table 1
Machine Learning (ML) & Metaheuristics (MH) Interactions.

Category	ML Technique/Procedure	MH Algorithm	Ref.
Low Level	db-scan	PSO	[14]
	Chess Rating	EA;GSA;ABC;DE	[37]
	Iterated Racing	ACO	[18]
	Decision Trees	GLS	[19]
	ID3	TS	[20]
	OBL	HS	[38]
	RL, DQN	ODD;FIFO;Flow Factor	[21]
	DQN	EA	[22]
	K-means	CS	[23]
High Level	Cooperative	DIMB	[24]
	Algorithm Selection	EA;Swarm intelligence	[27]
	RL	Randomized	[28]
	Perceptron	HyperHeuristics	[29]
	Q-learning RL	HyperHeuristics	[30]
	Greedy Learning	NEH;Several others	[31]
	Opportunity-Ordered Selection and KNN	SA;GA;GRASP;VNS:LNS	[25]
	Perceptron	MMOEAD15;HMPSO16; HMOEA06;MOGA06	[26]
Optimization Problem	Kriging	Simulated Annealing	[32]
	Neural Networks	NSGA-II	[33]
	Neural Networks	Multi-Objective Genetic	[34]
	Neural Networks	PSO	[35]
	Decision Trees	PSO	[36]

ML Induction Decision tree (ID3);Opposition-Based Learning (OBL); Reinforcement Learning (RL); Deep Q-Networks (DQN);K-Nearest Neighborhood (KNN)
MH Evolutionary Algorithm (EA);Ant Colony Optimization (ACO);Harmony Search (HS); Tabu Search (TB);Particle Swarm Optimization (PSO); Diversification–Intensification Memory Based (DIMB); Cuckoo Search CS.

3. The multidimensional knapsack problem

The multidimensional knapsack problem (MKP, [39]), corresponds to a non-deterministic polynomial-time ($\mathcal{NP}$)-hard combinatorial problem that considers multiple resource constraints, [40]. 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.

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

subject to:

$$\sum_{j=1}^{n} c_{ij} x_j \leq b_i, i \in \{1, \ldots, m\}. \tag{2}$$

$$x_j \in \{0, 1\}. \tag{3}$$

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

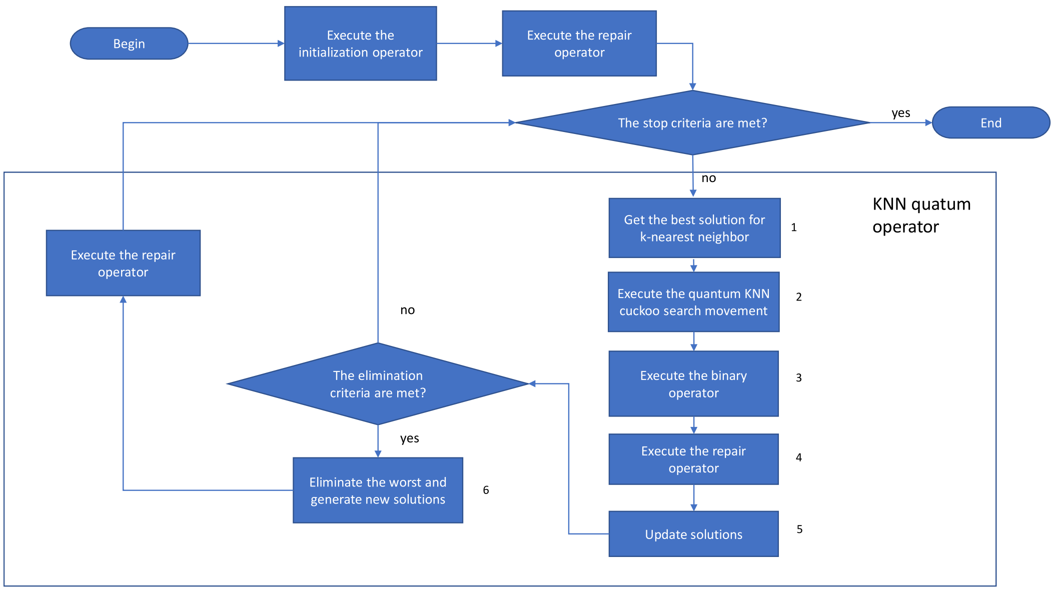
Begin
Execute the initialization operator
Execute the repair operator
The stop criteria are met?
yes
End
no
KNN quatum operator
Get the best solution for k-nearest neighbor
1
Execute the quantum KNN cuckoo search movement
2
Execute the binary operator
3
Execute the repair operator
4
Update solutions
5
Execute the repair operator
no
The elimination criteria are met?
yes
Eliminate the worst and generate new solutions
6

Fig. 1. A KNN quantum cuckoo search algorithm flow chart.

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.

MKP stands out for being a problem addressed by a large number of algorithms and, even so, it is still a very interesting challenge. This makes MKP a suitable problem for evaluating our hybrid algorithm. On the other hand, since it has a large number of applications, obtaining algorithms that improve the quality of solutions and decrease execution times has a relevant utility.

As with any heuristics, the metaheuristics proposed for the MKP aims two main features: improve near best solutions and decrease convergence times. Mostly, metaheuristics try to balance the exploration and exploitation in the search space by operators controlling movements and velocity, and nature-inspired algorithms are one of the most used [41,42].

Exemplar MKP applications are broad including time [43], space [44] and resource [45] allocations as well as partitioning [46].

Algorithms used in the MKP, include exact and heuristic methods. Exact algorithms, based on branch and bound [47] are common where [48,49] have achieved best performance. Exact algorithms perform acceptable at instances where the number of elements $n \in \{250, 500\}$ and the number of constraints m are contained in $\{5, 10\}$. However, for these types of instances, heuristic algorithms are more suitable because of their convergence times.

Heuristic algorithms addressing the MKP are numerous. Hybrid strategies try to complement the aspect of movements of the solution with execution times and expert knowledge. For example, in [50] an efficient tabu search method was used in combinations with an oscillation strategy and surrogate information constraints for balancing intensification and diversification properties. Others combine a hybrid strategy of a local search method with a filter & fun (F&F) procedure [51]. Whereas than, physically inspired operations can be also found in combination with nature-inspired procedures. For example, a quantum particle swarm optimization (QPSO*) was used additionally integrated with a local search operator and repairs methods [52]. Additional variants of the basic formulation are continually improved to explore and adapt to problems. In [53] a three-ratio self-adaptive particle swarm optimization (3R-BPSO) was proposed to tackle the MKP, replacing the pseudo-utility ratio as PSO is running and a local search mechanism to improve the quality of the solutions. In [54], a perturbation operator using KNN allowed to improve the results of random perturbation operators when solving the MKP.

Similarly, binary versions of algorithms are also developed for the assessment of the MKP. For example, a binary gray wolf optimizer (BGWO) was developed in [55], by directly adapting the motion equations of the metaheuristics to allow working in binary search spaces plus an elitist mechanism to generate the solutions population. Likewise, a binary fruit fly optimization algorithm was proposed with 3 search processes besides a global diversification mechanism in [56]. Finally, in [57], a binary artificial algae algorithm (BAAA) uses a transfer function as a binarization mechanism; whereas, in [58], a binary differential search algorithm (BDS) is proposed. Altogether, some structural algorithmic backbone can be depicted [41,42].

4. KNN quantum cuckoo search algorithm

This Section aims to provide a detailed description of the proposed algorithm. First, since the cuckoo search cannot be applied to binary search spaces directly, this algorithm must be adapted in order to solve this kind of problems. A number of binary variants have been proposed, for example, in [59] they use sigmoid functions to perform the binarization process. In [60], the Apache spark distributed computing framework was used together with the k-means clustering technique to get binary versions of cuckoo search in order to solve the crew scheduling problem.

In [61], a binary algorithm based on the use of sigmoid functions was proposed and applied to the optimal phasor measurement unit placement problem.

Whereas that in [62] the k-means technique as a binarization method is used to solve the counterfort retaining walls problem.

Fig. 1 shows the general diagram of the algorithm. The first stage corresponds to the initiation of solutions which will be described in Section 4.2. Subsequently, the procedure asks if the stop requirements have been met, in the case that they have not been met, the quantum KNN operator is executed. The purpose of this operator is to generate a new probability matrix that will allow binary solutions to be generated. The detail of the KNN quantum operator will be detailed in Section 4.4. Once the probability matrix has been updated, the binary solutions are generated to later replace the solutions that improve fitness.

Additionally, a criterion for generating new solutions has been incorporated. This procedure consists of eliminating the worst solutions once a criterion $t = 40$ of iterations is satisfied. If the iteration is a multiple of t, the $p_a = 10$ percent of the worst solutions are replaced by new solutions obtained from the solution initialization operator. Finally, with the new solutions, we proceed to repair them. The repair operator will be described in Section 4.3.

4.1. Cuckoo search algorithm

The reproductive strategy phenomena observed in cuckoo species, which lay their eggs in the nests of other bird species, has inspired the CS algorithm. Such is the level of sophistication of cuckoo birds that in some cases even the colors and patterns of the eggs of the chosen host species are mimicked. In the analogy, an egg corresponds to a solution. The concept behind the analogy is to use the best solutions (cuckoos) to replace those that do not perform well. The CS algorithm uses three basic rules:

1. Each cuckoo lays one egg at a time and deposits its egg in a randomly chosen nest.
2. The nests with the best results, that is, with high-quality eggs, will be considered in the next generation.
3. The number of nests available is a fixed parameter. The egg laid by a cuckoo can be discovered by the host bird with a probability $p_a \in (0, 1)$

The algorithm pseudo-code is shown in procedure 1.

Algorithm 1 Cuckoo search algorithm

```
1:  Objective function f(x)
2:  Generate initial solutions of n host nests.
3:  while stop criterion are meet do
4:      Get a cuckoo randomly and replace using Lévy flights.
5:      Evaluate the fitness.
6:      Choose in a random way a nest j among n:
7:      if $f_i > f_j$ then
8:          replace the solution.
9:      end if
10:     portion $p_a$ of the worst nests are eliminated and new ones
        are created.
11:     keep best solutions.
12:     find the current best
13: end while
```

4.2. Solution initialization operator

Given an MKP instance, a solution s is represented by a binary n-dimensional vector. In this vector, if $s_j = 1$, it means that the element has been selected and if $s_j = 0$, the element was not selected. According to the previous definition, our search space is defined as follows:

$$\Omega = \{(s_1, s_2, s_3 \ldots, s_n) \setminus s_j \in \{0, 1\}, 1 \leq j \leq n\}$$

The solution initiation procedure considers as input parameters the number of elements (n) that the instance considers and the number of solutions (ns) to use. The procedure returns the arrays $QCS(n, ns)$ and $DCS(n, ns)$ both with (n, ns) dimensions. DCS represents the binary solutions obtained at the initiation. QCS corresponds to the probabilistic representation of the solutions through a quantum vector q^i. In this vector a single bit could take the value "0" or "1", [52,63]. Then a probabilistic solution $q = (q^i_1, q^i_2, \ldots, q^i_n)$ with $q^i_j \in [0, 1]$ and $q \in QCS$, is transformed into a binary vector through Eq. (4).

$$s^i_j = \begin{cases} 1 & \text{if } r^i_j < q^i_j, \quad \forall j = 1, \ldots, n \\ 0 & otherwise \end{cases} \tag{4}$$

As a first step, the matrix QCS with values between 0 and 1 is generated randomly. Subsequently, a random number r is generated for each $QCS(j, i)$, if $QCS(j, i) > r$ then $DCS(j, i) = 1$ otherwise $DCS(j, i) = 0$. Once the DCS matrix has been generated, each of the solutions must be verified and, if necessary, they must be repaired. The procedure is detailed in Algorithm 2.

Algorithm 2 Initiation operator.

```
1:  function: initSolutions
2:  input: $n$ (number of elements), $ns$ (number of solutions)
3:  return: $QCS$ $(n, ns)$ matrix, $DCS$ discrete $(n, ns)$ matrix
4:  for $j$ in range(0,ns): do
5:      for $i$ in range(0,n): do
6:          $QCS(j, i) =$ rand(0,1)
7:      end for
8:  end for
9:  for $i$ in range(0,ns): do
10:     for $j$ in range(0,n): do
11:         $r =$ rand(0,1)
12:         if $QCS(j, i) > r$ then
13:             DCS(j,i) = 1
14:         else
15:             DCS(j,i) = 0
16:         end if
17:     end for
18: end for
19: return $QCS$, $DCS$
```

4.3. Repair operator

The repair operator aims to repair solutions that do not satisfy some of the restrictions. There are two cases where a repair needs to be executed. The first case runs after the generation of new solutions. The second is done after the generation of binary solutions. The procedure begins with a validation of the constraints. If any of the constraints are not met, the solution enters the repair process. The repair comprises two stages. In the first stage, the elements are removed according to the metric

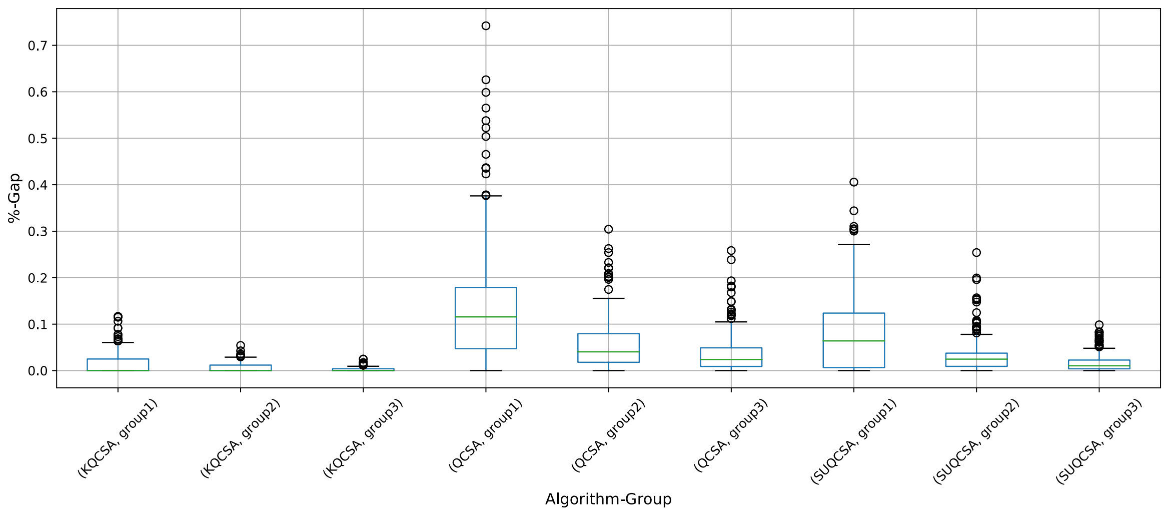

Fig. 2. % Gap box plots for 5–250 instances. Algorithms KQCSA, QCSA, and SUQCSA.

defined in Eq. (5) and proposed in [64].

$$\sigma_j = \frac{\sum_{i=1}^{m} \frac{c_{ij}}{mb_i}}{p_j} \tag{5}$$

The elements are removed considering σ_j from low to high until all constraints are satisfied. Subsequently, in the second stage, the incorporation of new elements is evaluated. For this incorporation, the elements are ordered from highest to lowest σ_j. If a selected element allows the solution to satisfy all the restrictions, then it is added to the solution.

Algorithm 3 Repair operator.

```
1: function: reapirSolution
2: input: $s$ (solution)
3: return: $s$ (repair solution)
4: if $s$ satisfy the constraints then
5:     return $s$
6: else
7:     while $s$ Don't Satisfy Constraints do
8:         $s$ = removeElement($s$)
9:     end while
10:    while $s$ Satisfy Constraints do
11:        $s$ = addElement($s$)
12:    end while
13:    return $s$
14: end if
```

4.4. KNN quantum operator

The KNN quantum operator is the main module of KQCSA. This operator aims to update the binary solutions by using the cuckoo search algorithm, a probability matrix, and a solution update method, based on the use of the k-nearest neighbors concept. Let $DCS^i(t)$ be a point in our search space, that is, a vector with values 0 and 1 that solves an instance of the knapsack problem. This solution was obtained in iteration t. Then, for each solution $DCS^i(t)$, the k-nearest neighbors of $DCS^i(t)$ are obtained, and using the k-neighbors, the best solution is selected. This procedure is carried out for each solution i, generating a matrix DCS_k, which corresponds to all k-best solutions. The previous procedure runs on line 4 of Algorithm 4 and to box 1 of Fig. 1.

The next step is to get the new probability matrix $QCS(t+1)$ from $QCS(t)$, $DCS_k(t)$ and $DCS(t)$. This is modeled by Eq. (6). Remember that, in the process of initiating solutions, an initial QCS is defined.

$$QCS(t+1) = \alpha * QCS(t) + (1-\alpha) * norm(nLevy(\lambda) \oplus (DCS_k(t) - DCS(t))) \tag{6}$$

Therefore, if we consider the component j of the solution i, the value of $QCS^i(t)_j$, models the probability of obtaining 1 in that component and solution. In this equation, $(DCS^i_k(t)_j - DCS^i(t)_j)$ can take the values $\{-1, 0, 1\}$. Subsequently, an entrywise multiplication is executed with the normalized vector generated by the cuckoo search metaheuristic. Finally, the list resulting from the operation $nLevy(\lambda) \oplus (DCS_k(t) - DCS(t))$, the min–max normalization is applied to bring the values between 0 and 1. The result obtained from normalization is scaled by $(1-\alpha)$, where $\alpha = 0.2$. Finally, the result of the above is added to the probability $QCS(t)$ scaled by the factor α. Once $QCS(t+1)$ has been obtained. This corresponds to box 2 of Fig. 1 and to line 5 of the Algorithm 4.

Here we must observe that Algorithm 2, uses Eq. (4). In this equation, for the case that the inequality $r^i_j < q^i_j$, the value 1 is assigned. The definition of this rule is based on Eq. (6). Suppose that $DCS^i_k(t)_j$ has the value 1 and $DCS^i(t)_j$ has the value 0, then $DCS^i_k(t)_j - DCS^i(t)_j$ takes the value 1. On the other hand, if $DCS^i_k(t)_j$ has the value 0 and $DCS^i(t)_j$ has the value 1, then $DCS^i_k(t)_j - DCS^i(t)_j$ takes the value -1. So when multiplying this factor by the normalized values of $nLevy$ and then performing the normalization, we obtain values greater than q^i_j in the first case, which is consistent with moving $DCS^i(t)_j$ from 0 to 1.

The next step corresponds to getting the binary solutions using the procedure detailed in Algorithm 2. This procedure corresponds to line 6 of Algorithm 4 and box 3 of Fig. 1. After the binary solutions are obtained, they are repaired if necessary. Once the solutions are repaired, a KNN solution update procedure, shown in box 5 of Fig. 1, is carried out. In this procedure, first is asked if the fitness of the solution s^i is worse than the current worst fitness (worstFitness), then the solution is discarded. Otherwise, then it is evaluated if the fitness of s^i is better than the fitness of the current solution ($DCS^i(t)$), if the criterion is met, the solution is updated. In the case that it is not fulfilled, the condition, a

Table 2
Parameter setting for the Cuckoo search algorithm.

Parameters	Description	Value	Range
N	Number of Nest	120	[100, 120, 140]
α	Probability matrix update coefficient	0.1	[0.1, 0.2, 0.3]
K	Neighbors number	10	[10, 15, 20]
γ	Step Length	0.01	0.01
κ	Levy distribution parameter	1.5	1.5
Iteration Number	Maximum iterations	1200	[1000, 1200, 1500]

random number is obtained and compares it with a threshold ϕ together with obtaining the worst solution of the k nearest neighbors ($worstnbr$). In case the threshold condition is met and the value of the solution is better than $worstnbr$, then the worst k-nearest neighbor solution is replaced. Finally, every $t = 40$ iterations, a procedure is carried out, which corresponds to box 6 of Fig. 1, with the aim of eliminating the percentage $p_a = 10\%$ of the worst solutions. The detail of the KNN quantum operator is shown in Algorithm 4.

Algorithm 4 Quantum KNN operator.

```
1: function: QKNNCS
2: input: $QCS$ $(n, ns)$ matrix, $DCS$ discrete $(n, ns)$ matrix
3: return: $QCS$ $(n, ns)$ matrix, $DCS$ discrete $(n, ns)$ matrix
4: $DCS_k(t)$ = getDCSk(DCS(t),k)
5: $QCS(t+1) = \alpha * QCS_(t) + (1-\alpha) * normLevy(\lambda) \oplus (DCS_k(t) - DCS(t))$
6: $DCS^*(t+1)$ = getBinarySolutions($QCS(t+1)$)
7: $DCS^*(t+1)$ = repairSolution($DCS^*(t+1)$)
8: for $s^i$ in $DCS^*(t+1)$ do
9:   if $s^i$.fitness $<$ worstFitness then
10:     don't update
11:   else if $s^i$.fitness $>$ $DCS^i(t)$.fitness then
12:     $DCS^i(t) = s^i$
13:   else
14:     $nbr$ = getkNeighbor($DCS^i(t)$,$k$)
15:     $r$ = rand(0,1)
16:     if $r > \phi$ and $s^i$.fitness $>$ $worstnbr$.fitness then
17:       $DCS(t)(worstnbr) = s^i$
18:     end if
19:   end if
20: end for
21: $DCS(t+1) = DCS(t)$
22: if $iteration$ mod $t$ == 0 then
23:   $newSols$ = generateNewSolutions($p_a$)
24:   $newSols$ = repairSolutions($newSols$)
25:   $DCS(t+1)$ = updateNewSolutions($newSols$, $DCS(t+1)$)
26: end if
```

5. Results

This section details the experiments carried out with KQCSA aimed to evaluate the value and contribution of the proposed algorithm when is applied to an $\mathcal{NP}$-hard combinatorial problem. The MKP was chosen as a problem to use since it is a problem that has been addressed by various algorithms, uses particularly large instances, and it is not trivial to obtain good performance. However, we must emphasize that the neighborhood technique used in KQCSA can be easily adapted to other optimization algorithms. CS was used as the optimization algorithm since it is a simple algorithm to parameterize and it has been used in solving various optimization problems. Here we must note that the proposed equation can also be adapted to other optimization algorithms.

In Section 5.1, the methodology applied to get the value of the parameters used by KQCSA will be detailed. After that, in Section 5.2, we will study the contribution of the quantum operator and the designed KNN solution update procedure, which is based on the concept of k-nearest neighbors. Finally, in Section 5.3, comparisons will be made with state-of-the-art algorithms in order to evaluate the efficiency of our proposal. Instances containing 250 and 500 elements of the OR-library were selected to perform the experiments. These instances correspond to the most difficult instances of the library.

In the construction of the algorithm, Python 3.6, and a PC with Windows 10, a core i7 processor, and 16GB in RAM were used. The Wilcoxon signed-rank [65] method was the test selected to determine if the difference is statistically significant. The p-value used was 0.05. The selection of the test is based on the methodology proposed in [66]. In this methodology, the Shapiro–Wilk normality test is applied first. In the event that one of the populations is not normal, and both populations have the same number of points, the Wilcoxon signed-rank is suggested to verify the difference. In our case, the Wilcoxon test was applied to the entire population of instances considering all executions and comparing the results of both algorithms. Each instance was resolved 30 times and the best value and averages indicators were obtained. In addition, the average time it took for the algorithm to get the best solution in each run is reported. The OR-Library dataset is share in,[1] and the details of the results are shared in.[2]

5.1. Parameter setting

In the selection of parameters, the methodology proposed in [67] was used. To obtain an adequate selection of the parameters, this methodology uses 4 measures defined by the Eqs. (7) to (10). For the generation of values, problems 0, 10, and 20 of the instances mkp.5.250, mkp.10.250, and mkp.30.250 were chosen. Each combination of parameters was executed 10 times. The set of explored and selected parameters are shown in Table 2.

[1] OR-Library: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html.
[2] Results: https://drive.google.com/drive/folders/1DEsvAE1DN63z1Fis3xjwiWqpn-07WPlf?usp=sharing.

1. The percentage deviation of the best value obtained compared to the best known value:

$$bSolution = 1 - \frac{KnownBestValue - BestValue}{KnownBestValue} \tag{7}$$

2. The percentage deviation of the worst value obtained compared to the best known value:

$$wSolution = 1 - \frac{KnownBestValue - WorstValue}{KnownBestValue} \tag{8}$$

3. The percentage deviation of the average value obtained compared to the best known value:

$$aSolution = 1 - \frac{KnownBestValue - AverageValue}{KnownBestValue} \tag{9}$$

4. The convergence time for the best value:

$$nTime = 1 - \frac{convergenceTime - minTime}{maxTime - minTime} \tag{10}$$

5.2. Insight into KNN quantum algorithm

The objective of this section is to determine the contribution of the quantum KNN operator and the KNN solution update procedure based on the concept of the k-nearest neighbor, in the final result. To address this challenge, two algorithms are designed. The first algorithm is a quantum cuckoo search algorithm (QCSA).

QCSA has the same operators as KQCSA illustrated in Fig. 1, except the KNN quantum cuckoo search operator, which is replaced by a standard quantum operator. This standard operator is defined in Eq. (11). Comparing this Equation with Eq. (6), it is observed that the best k-nearest neighbor solution $DCS_k(t)$ is replaced by the best global solution obtained up to iteration t. This best solution will be denoted by $BDCS(t)$. With this replacement, it is being evaluated the contribution of using the best neighbor k ($DCS_k(t)$) in Eq. (6). It is important to mention that QCSA does not use the KNN solution update procedure, but rather, the update procedure executed in line 12 of Algorithm 4.

The second algorithm aims to evaluate the contribution of the KNN solution update procedure. To carry out the evaluation, an alternative update procedure is built, the standard update quantum cuckoo search algorithm (SUQCSA). KQCSA uses a solution update procedure based on the k-nearest neighbor. This procedure is detailed on lines 9 through 18 of the Algorithm 4, in Section 4. Specifically, between lines 16 to 18 of this algorithm, the updating of a new solution s^i is allowed, considering the neighbors of s^i. In the case of SUQCSA, it uses the standard update defined in line 12. Therefore, the update based on neighboring neighbors is not executed, so lines 13 to 17 are not executed.

$$QCS(t+1) = \alpha * QCS(t) + (1-\alpha) * norm(nLevy(\lambda) \oplus (BDCS(t) - DCS(t))) \tag{11}$$

For the execution of these experiments, the problems 5.250, 10.250, and 30.250 were used. Where group 1 corresponds to problems from 0 to 9, group 2, from 10 to 19, and group 3, from 20 to 29. The results are shown in Tables 3 to 5. In Table 3, the best, and the average values for each of the used instances are displayed for each algorithm.

For the comparisons, all three algorithms perform similarly in obtaining the best values. Although, the Wilcoxon test shows $p = 0.068$ for the comparison between KQCSA and QCSA, indicating no significant differences in finding the best solutions. However, when analyzing the average, the situation is completely different. KQCSA performs significantly better for all problems. When we analyze the results of the box plots and their interquartile ranges, in Fig. 2, we observe that KQCSA improves in lowering variance, but also decreasing outset distribution, at a time of improving convergence toward the best-known values for the three difficulty groups.

From Figs. 3, and 4, we conclude a clear gain in performance for solving KMP with KQCSA, with a significant decrease in the variance of solutions as well as lowering the difference with best-known solutions. Moreover, KQSCA algorithm scales with the difficulty of the problem better than the other algorithms. Next, when analyzing the convergence charts, it is observed that KQCSA performs better than SUQCSA and QCSA. Faster rates of convergence are observed in favor of KQCSA in all cases. Moreover, the final quality of the solution is improved in all cases.

The results obtained with the data set 10.250, shown in Table 4 and Figs. 5, 6, and 7, are similar to those described in the previous case. The best value indicator is very similar to the three algorithms and there is a significant difference in the case of the average indicator, in which KQCSA always obtains the best values. When analyzing the box plots, these show a lower dispersion and better values for KQCSA. Finally, in the case of convergence charts, KQCSA shows better convergence in the three groups.

In the case of the data set 30.250, shown in Table 5 and Figs. 8, 9, and 10, differs from the previous ones. For this data set, the best value indicator shows significant differences in favor of KQCSA. In 16 of the 30 problems, KQCSA obtained better values than SUQCSA and QCSA. The average indicator, box plots, and convergence charts, KQCSA was superior in the 3 groups of problems.

KQCSA consistently performed better than QCSA and SUQCSA in all instances. In the case of the comparison with QCSA, the objective was to identify the contribution of integrating the k-nearest neighbors in the CS movement, defined in Eq. (6). The previous results indicate that this integration contributes to obtaining better solution values and convergences. In the case of the comparison of KQCSA with SUQCSA, the objective was to evaluate the update procedure carried out in line 17 of Algorithm 4. The results show that the update procedure improves the quality of the solutions obtained.

Finally, the times used in the KNN fit and the execution times spent in obtaining the nearest neighbors were analyzed. The data set corresponds to 120 solutions with a dimension of 250. The times were measured in each iteration and their distributions are shown in Figs. 11 and 12. Fig. 11 shows the distribution of the fit times, observing that most of the time these are less than 0.5 ms. In the case of the times used to obtain the nearest neighbors, the distribution in Fig. 12 represents the times used to obtain the neighbors of all solutions. In the Figure, it is observed that most of the time these are less than 5 ms.

5.3. Comparisons

This section compares the performance of KQCSA against different state-of-the-art algorithms that have solved the MKP. In general, comparison aims to find the best performing algorithm from the pool in terms of optimal solutions, scalability, and execution times. We considered the best solution found for each algorithm for different running instances and the average of the solutions found. We test performance on the pool of MKP problems with 5, 10, and 30 constraints, for problem set with 500 items; broadly known as *mkp*5.500, *mkp*10.500, and *mpk*30.500 respectively.

As instances used in the evaluation, *mkp*.5.500, *mkp*.10.500 and *mkp*.30.500 were considered, we compared against different pools of algorithms previously evaluated in the literature: mostly corresponding to the binary extension of nature-inspired algorithms (AAA, GWO, PSO), including several variants of one of the

Table 3
OR-Library benchmarks MKP, *mkp.5.250*.

Instance	Best Known	SUQCSA Best	SUQCSA Avg	SUQCSA time(s)	QCSA Best	QCSA Avg	QCSA time(s)	KQCSA Best	KQCSA Avg	KQCSA time(s)
5.250.0	59312	59312	59257.66	59.9	59312	59200.14	56.3	59312	**59300.31**	46.1
5.250.1	61472	61472	61438.62	47.7	61472	61417.38	46.4	61472	**61469.79**	36.7
5.250.2	62130	62130	62064.72	53.6	62130	61991.62	51.2	62130	**62130.00**	41.2
5.250.3	59463	59446	59419.03	56.3	59446	59404.45	52.8	59446	**59433.79**	43.3
5.250.4	58951	58951	58896.72	66.6	58951	58875.10	61.9	58951	**58950.69**	51.2
5.250.5	60077	60061	60022.38	62.7	60062	60009.69	61.3	60062	**60048.69**	48.2
5.250.6	60414	60414	60372.03	48.2	60414	60346.14	45.3	60414	**60414.00**	37.1
5.250.7	61472	61472	61449.66	48.2	61472	61420.38	46.7	61472	**61464.52**	38.2
5.250.8	61885	61885	61859.69	49.7	61885	61819.31	43.4	61885	**61879.00**	34.2
5.250.9	58959	58959	58858.76	44.5	58959	58810.03	40.8	58959	**58957.41**	32.3
5.250.10	109109	109109	109079.00	42.0	109109	109032.24	43.8	109109	**109100.31**	34.2
5.250.11	109841	109841	109779.28	44.5	109841	109758.07	45.8	109841	**109840.48**	37.1
5.250.12	108508	108489	108474.93	44.5	108489	108438.28	50.9	**108508**	**108505.59**	41.3
5.250.13	109383	109383	109356.76	48.2	109383	109336.03	47.6	109383	**109380.03**	37.5
5.250.14	110720	110718	110686.59	53.7	110718	110664.07	46.4	110718	**110716.21**	38.2
5.250.15	110256	110243	110213.03	48.8	110256	110190.62	45.5	110256	**110234.03**	37.9
5.250.16	109040	109040	109009.28	49.7	109040	108988.97	40.8	109040	**109036.34**	33.2
5.250.17	109042	109042	109021.72	49.3	109042	109003.41	51.5	109042	**109038.90**	42.4
5.250.18	109971	109971	109946.28	43.2	109971	109904.41	49.3	109971	**109970.17**	39.2
5.250.19	107058	107038	107021.45	55.1	107058	107006.90	36.7	107058	**107041.31**	30.1
5.250.20	149665	149659	149641.66	51.0	149638	149625.55	41.9	**149665**	**149658.52**	33.2
5.250.21	155944	155940	155928.34	39.1	155940	155881.03	44.3	155940	**155940.00**	34.5
5.250.22	149334	149316	149299.66	43.2	149316	149295.72	47.7	**149334**	**149314.86**	38.1
5.250.23	152130	152130	152086.21	44.9	152130	152057.38	46.5	152130	**152130.00**	37.5
5.250.24	150353	150353	150342.10	49.5	150353	150309.72	46.6	150353	**150353.00**	36.8
5.250.25	150045	150045	149975.34	48.8	150045	149944.07	47.2	150045	**150045.00**	37.4
5.250.26	148607	148607	148577.90	47.8	148607	148533.86	41.2	148607	**148604.41**	32.8
5.250.27	149782	149772	149763.24	48.6	149772	149731.90	43.8	149772	**149771.86**	34.2
5.250.28	155075	155075	155064.03	42.6	155075	155039.10	39.9	155075	**155071.14**	32.5
5.250.29	154668	154662	154659.76	44.5	154662	154635.93	44.2	**154668**	**154666.97**	35.2
Average	107088.87	107084.50	107052.19	49.1	107084.93	107022.38	46.9	**107087.27**	**107082.24**	37.7
Wilcoxon *p*-value		0.018	1.73e−06		0.068	1.23e−07				

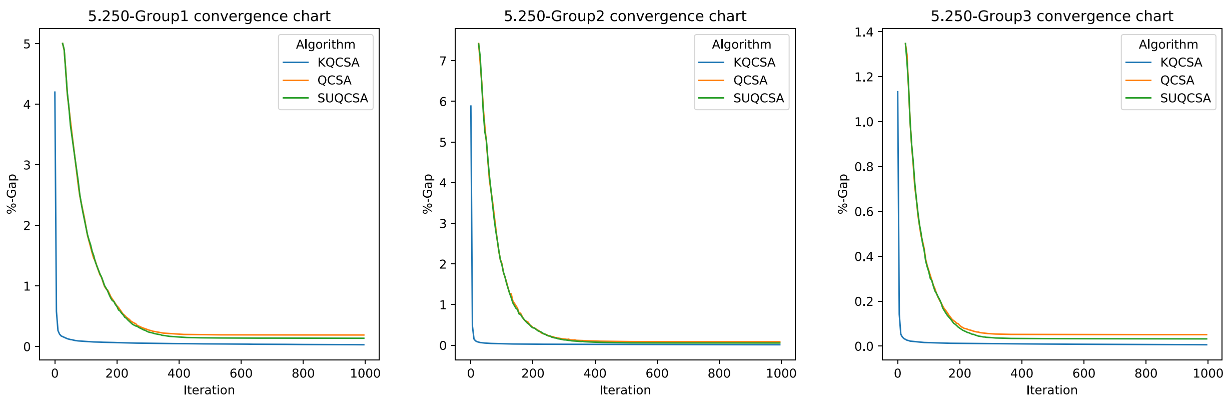

Fig. 3. % Gap convergence chart for 5–250 instances.

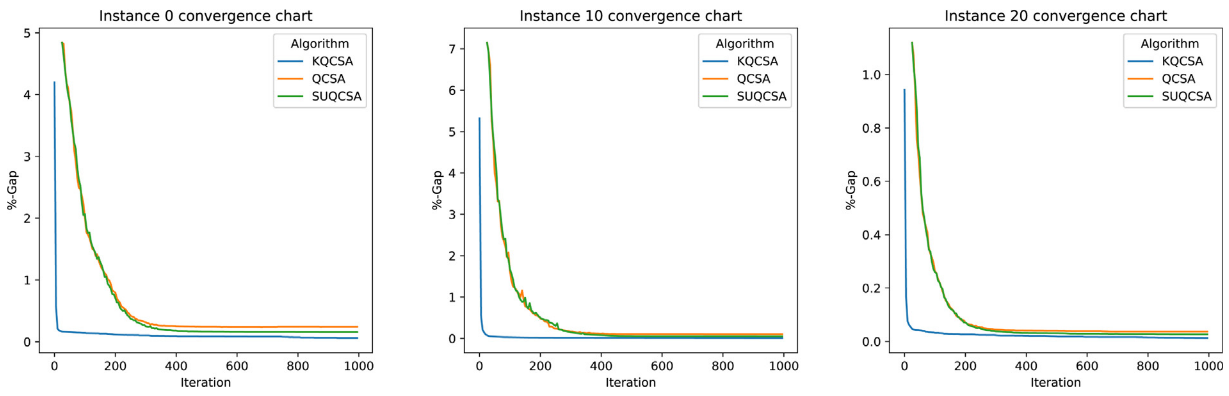

Fig. 4. % Gap convergence chart for 0, 10 an 20 instances of 5–250 problems.

Table 4
OR-Library benchmarks MKP *mkp.10.250*.

Instance	Best Known	SUQCSA B est	SUQCSA Avg	SUQCSA time(s)	QCSA Best	QCSA Avg	QCSA time(s)	KQCSA Best	KQCSA Avg	KQCSA time(s)
10.250.0	59187	59187	59142.18	125.7	59187	59099.32	122.1	59187	**59184.61**	96.7
10.250.1	58781	58685	58644.86	137.9	58685	58606.76	131.7	**58705**	**58682.31**	106.1
10.250.2	58097	58056	58017.03	121.4	58094	58013.41	113.0	58094	**58087.97**	93.4
10.250.3	61000	60972	60919.59	126.4	60946	60893.41	123.4	**61000**	**60943.24**	97.2
10.250.4	58092	58092	58011.52	107.2	58092	57966.72	102.7	58092	**58041.90**	82.5
10.250.5	58824	58803	58725.72	102.8	58803	58689.10	101.4	58803	**58786.17**	79.1
10.250.6	58704	58632	58571.93	133.2	58704	58529.93	128.1	58704	**58686.52**	102.5
10.250.7	58936	58917	58865.76	102.8	58930	58842.24	97.4	**58936**	**58910.59**	79.1
10.250.8	59387	59382	59361.48	122.6	59382	59323.07	115.5	**59384**	**59381.97**	94.3
10.250.9	59208	59208	59104.24	118.8	59208	59047.28	114.5	59208	**59201.28**	91.4
10.250.10	110913	110913	110874.24	106.6	110913	110822.59	100.6	110913	**110911.28**	82,6
10.250.11	108717	108717	108638.32	97.8	108717	108580.47	97.4	108717	**108700.86**	77.4
10.250.12	108932	108932	108883.03	100.6	108932	108854.14	98.4	108932	**108915.76**	79.1
10.250.13	110086	110086	110015.76	102.8	110086	109992.14	100.5	110086	**110051.55**	73.2
10.250.14	108485	108485	108441.72	95.2	108485	108391.07	92.5	108485	**108483.14**	84.7
10.250.15	110845	110841	110785.45	110.1	110829	110726.38	108.5	110841	**110816.90**	81.8
10.250.16	106077	106077	106063.31	106.3	106077	106046.97	99.8	106077	**106076.45**	79.3
10.250.17	106686	106686	106662.07	103.1	106686	106633.62	98.3	106686	**106686.00**	75.2
10.250.18	109829	109821	109781.00	97.8	109825	109762.93	95.4	109825	**109814.34**	73.7
10.250.19	106723	106723	106657.48	95.8	106723	106612.79	94.5	106723	**106706.48**	84.2
10.250.20	151809	151809	151773.69	109.5	151801	151751.55	107.8	151809	**151790.03**	64.7
10.250.21	148772	148772	148696.00	84.1	148772	148700.76	81.0	148772	**148772.00**	67.9
10.250.22	151909	151909	151885.59	88.3	151909	151827.62	85.4	151909	**151901.90**	66.2
10.250.23	151324	151256	151236.07	86.1	151275	151201.97	80.0	151275	**151258.55**	74.7
10.250.24	151966	151948	151902.21	97.1	151948	151883.97	92.3	151948	**151934.83**	73.2
10.250.25	152109	152109	152076.97	95.2	152109	152049.86	92.2	152109	**152109.00**	68.1
10.250.26	153131	153131	153080.93	88.5	153131	153018.28	84.4	153131	**153129.45**	66.2
10.250.27	153578	153578	153500.72	86.1	153520	153480.90	79.8	153578	**153528.55**	61.4
10.250.28	149160	149155	149108.00	79.8	149130	149079.21	76.0	149155	**149134.41**	70.4
10.250.29	149704	149704	149664.21	91.5	149704	149628.76	86.4	149704	**149698.31**	72.3
Average	106365.7	106352.87	106303.04	100.8	106353.43	106268.57	96.9	**106359.60**	**106344.21**	77.5
Wilcoxon *p*-value		0.011	1.31e−6		0.012	1.54e−7				

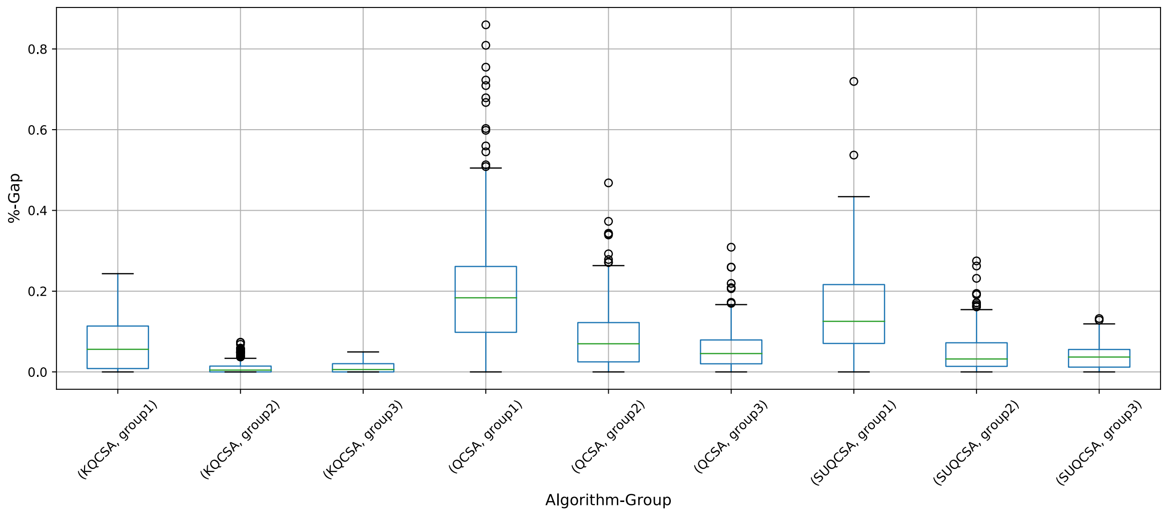

Fig. 5. % Gap box plots for 10–250 instances. Algorithms KQCSA, QCSA, and SUQCSA.

best known and performant as it is particle swarm optimization (PSO).

For the *mkp*.5.500 problem set, the algorithms chosen were: the Binary Artificial Algae Algorithm (BAAA) [57], which uses a binary mechanism based on transfer functions; QPSO*, [52], which corresponds to a quantum binary version of PSO; and 3R-BPSO [53], which implements a three pseudo-utility radio self-adaptive method together with a binary version of PSO.

The main reason for choosing these three algorithms is because they use swarm intelligence that naturally work in continuous space and therefore have been adapted through three different strategies to be able to tackle combinatorial problems. The BAAA algorithm uses a general binarization mechanism, that is, it does not modify the equations of motion, [68], by applying transfer functions. These transfer functions consider BAAA transitions and translate them into transition probabilities. These probabilities are subsequently binarized. Additionally, BAAA uses a relative mean resource occupation ratio, equivalent to the one used in Eq. (5) for its initialization and repair process.

In the case of 3R-PSO, it also uses the transfer functions method to perform binarization, however, instead of using Eq. (5) within its heuristic, it uses three ratios: surrogate relaxation ratio, profit density, and profit/weight utility. With this variation, it allows PSO solutions to generate a better exploration of the

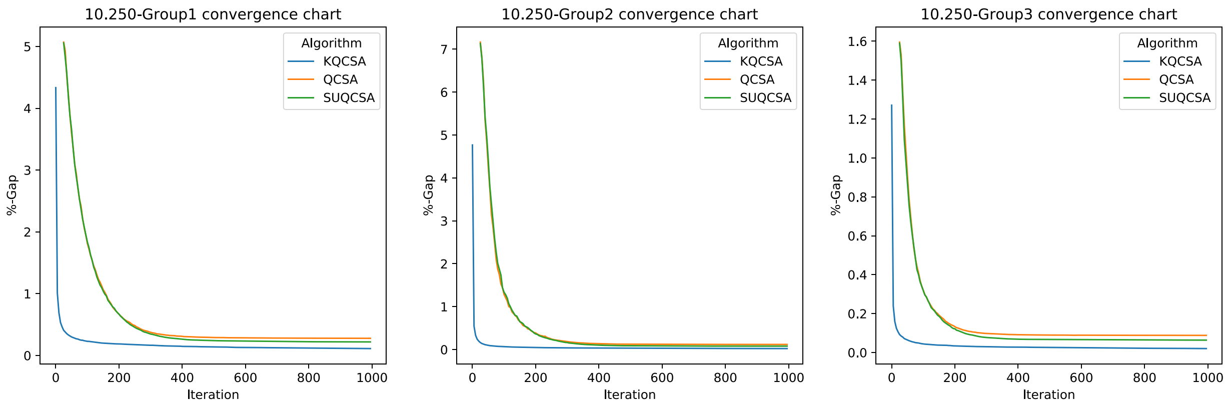

Fig. 6. % Gap convergence chart for 10–250 instances.

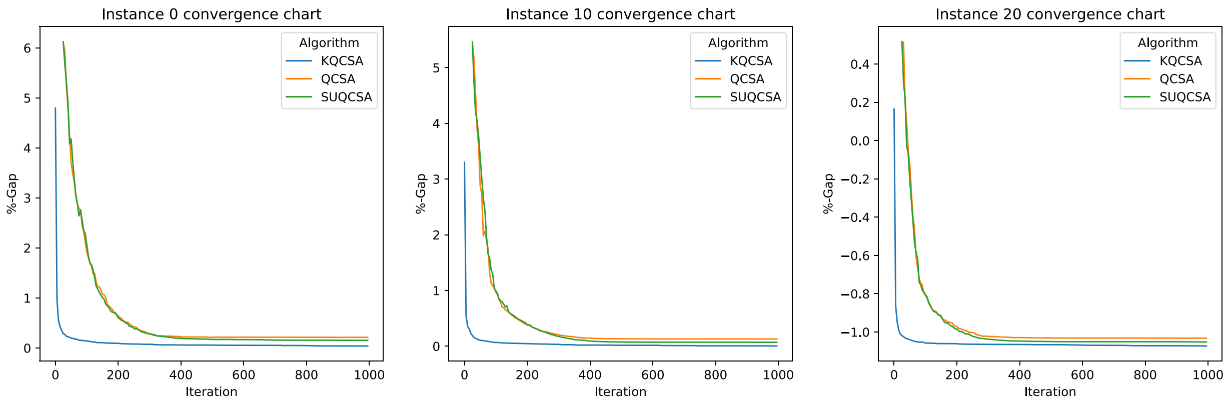

Fig. 7. % Gap convergence chart for 0, 10 an 20 instances of 10–250 problems.

Table 5
OR-Library benchmarks MKP *mkp.30.250*.

Instance	Best Known	SUQCSA B est	SUQCSA Avg	SUQCSA time(s)	QCSA Best	QCSA Avg	QCSA time(s)	KQCSA Best	KQCSA Avg	KQCSA time(s)
30.250.0	56842	56702	56639.83	165.5	56702	56587.86	156.8	**56796**	**56755.66**	127.3
30.250.1	58520	58318	58241.00	187.6	58315	58183.28	184.1	**58520**	**58487.52**	144.3
30.250.2	56614	56515	56394.90	319.9	56418	56296.28	297.4	**56614**	**56579.31**	246.1
30.250.3	56930	56865	56756.62	392.1	56865	56716.24	378.6	**56930**	**56896.66**	301.6
30.250.4	56629	56629	56541.34	442.6	56629	56472.31	428.6	56629	**56610.28**	340.5
30.250.5	57205	57136	57020.79	305.0	57136	56950.83	291.4	**57149**	**57118.93**	234.6
30.250.6	56348	56290	56088.55	237.2	56166	56053.79	224.9	**56303**	**56237.55**	182.5
30.250.7	56457	56392	56279.66	477.5	56392	56234.55	464.0	56392	**56351.34**	367.3
30.250.8	57474	57341	57198.07	424.3	57429	57188.10	412.9	**57474**	**57407.55**	326.4
30.250.9	56447	56447	56242.34	391.9	56447	56164.31	382.6	56447	**56325.00**	301.5
30.250.10	107770	107703	107623.83	312.4	107689	107575.52	305.0	**107712**	**107683.66**	240.3
30.250.11	108392	108332	108180.83	300.8	108379	108146.00	284.0	108379	108343.72	231.4
30.250.12	106442	106392	106299.21	320.7	106392	106267.52	299.1	106392	**106364.24**	246.7
30.250.13	106876	106733	106661.34	261.9	106733	106607.59	243.2	**106851**	**106816.93**	201.5
30.250.14	107414	107396	107280.07	281.8	107396	107235.07	266.2	107396	**107352.03**	216.8
30.250.15	107271	107202	107053.14	391.8	107202	107038.52	378.7	**107271**	**107353.90**	301.4
30.250.16	106372	106308	106168.62	349.1	106223	106115.97	331.1	106308	**106218.41**	268.5
30.250.17	104032	103985	103887.93	306.4	103974	103861.79	293.4	103985	**103951.24**	235.7
30.250.18	106856	106800	106690.93	273.1	106800	106646.00	259.4	**106835**	**106803.34**	210.1
30.250.19	105780	105751	105594.86	268.8	105700	105555.59	259.9	105751	**105690.93**	206.8
30.250.20	150163	150019	149997.97	240.9	150019	149959.79	223.9	**150082**	**150035.55**	185.3
30.250.21	149958	149907	149848.38	217.2	149907	149798.38	209.9	149907	**149897.66**	167.1
30.250.22	153007	152993	152929.45	232.6	152993	152869.97	221.1	152993	**152966.79**	178.9
30.250.23	153234	153173	153082.62	267.5	153175	153033.97	259.0	**153234**	**153207.28**	205.8
30.250.24	150287	150287	150200.76	293.3	150287	150128.38	279.0	150287	**150278.72**	225.6
30.250.25	148574	148553	148431.31	225.2	148553	148392.55	216.7	148553	**148532.66**	173.2
30.250.26	147477	147477	147410.83	203.8	147477	147376.28	200.2	147477	**147469.17**	156.8
30.250.27	152912	152852	152750.17	189.9	152852	152711.52	182.4	**152912**	**152879.31**	146.1
30.250.28	149570	149568	149482.97	254.0	149568	149427.45	236.6	**149570**	**149553.97**	195.4
30.250.29	149668	149601	149500.72	249.7	149601	149448.41	244.6	**149668**	**149647.31**	192.1
Average	104717.37	104657.67	104549.30	292.8	104647.30	104501.46	280.5	104698.00	104660.55	225.3
Wilcoxon *p*-value		0.011	1.84e−5		0.012	2.11e−5				

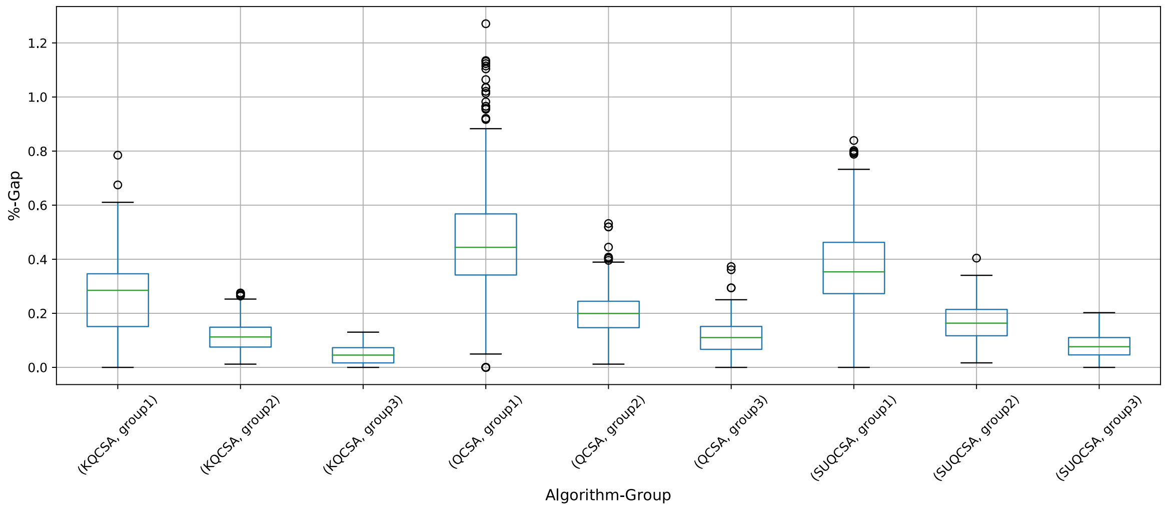

Fig. 8. % Gap box plots for 30–250 instances. Algorithms KQCSA, QCSA, and SUQCSA.

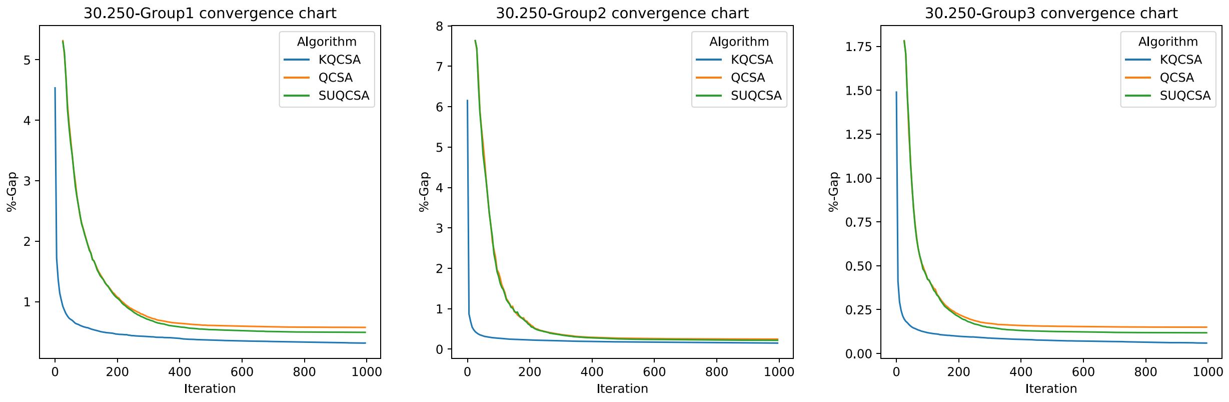

Fig. 9. % Gap convergence chart for 30–250 instances.

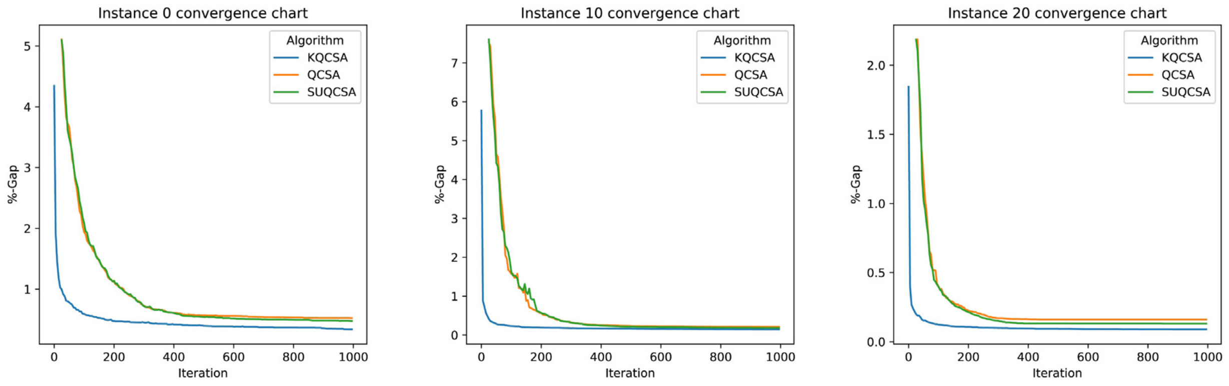

Fig. 10. % Gap convergence chart for 0, 10 an 20 instances of 30–250 problems.

search space. Finally, QPSO* is a specific binarization mechanism, that is, unlike the previous two cases; QPSO* adapts the Equations of the metaheuristic to be able to address the combinatorial problem, [68]. Additionally, QPSO* in its repair process uses Pseudo-utility ratio.

The results are detailed in Table 6. When analyzing the best value indicator, we observe that 3R-PSO obtained 13 best values, QPSO*, 15 best values, and KQCSA 25. We must note that when two algorithms obtained the best value, we counted it in both, for this reason, the total sum is greater than 30. When the distribution of best values is analyzed, we see in the three groups 0, 1, and 2, KQCSA is superior to the other algorithms, the superiority being measured by the Wilcoxon test. Therefore, KQCSA gets significantly better best values. The next step is to see if this behavior is consistent across all executions. For this, the average indicator will be analyzed. In the average of the best values, KQCSA obtained the best result. Wilcoxon test indicated that the difference is statistically significant. In the case of the average indicator, KQCSA was superior in all instances, in addition to obtaining small values in the standard deviation.

In Table 7, the results of the experiments carried out with *mkp*.10, 500 are shown. In these experiments, the 3R-BPSO and

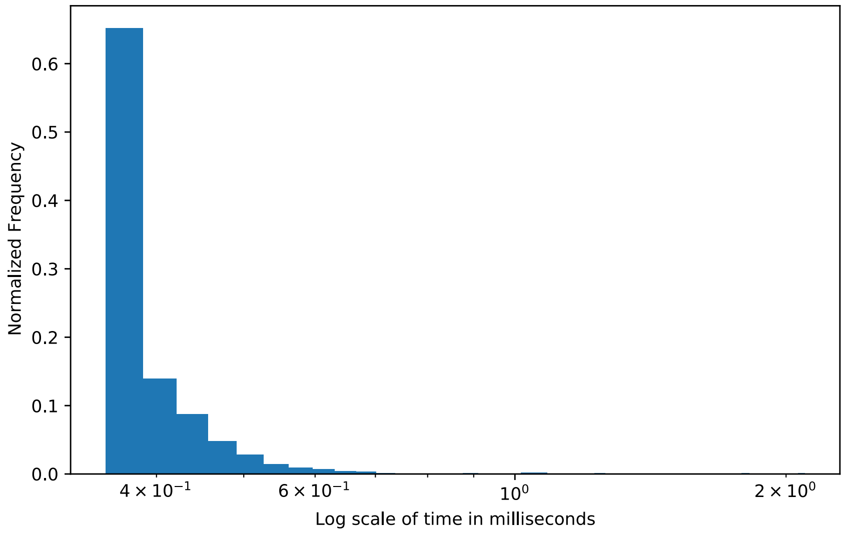

Fig. 11. KNN training histogram time for each iteration for 250 instances.

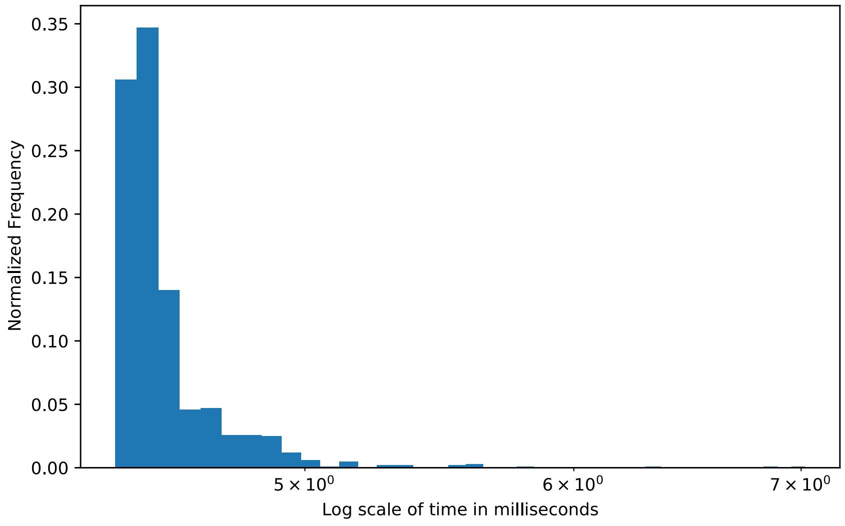

Fig. 12. KNN runtime histogram considering all solutions for each iteration on 250 instances.

QPSO* algorithms, already used in the previous case, were considered and, in addition, TE-DBS, [58], which uses a binarization mechanism based on transfer functions, similar to that used in the BAAA algorithm. Additionally, F & F, [51] was included, where only the best values are reported. BAAA was not considered because no results are reported with this data set.

When we analyze the best value indicator, we observe that F& F obtained 1 best value, 3R-BPSO, 5 best values, and KQPSO 10 best values. Later QPSO* and TE-DBS obtained 12 and 15 best values, respectively. However, the total average of the best values, KQCSA obtained the best result, and the Wilcoxon test indicated that the difference was significant in all cases. This last result indicates that KQCSA consistently gets better best values than the other algorithms. In the case of the average indicator, TD-DBS obtained the best result in 1 instance, QPSO* in 8 instances, and finally, KQCSA in 21 instances. The Wilcoxon test again shows that the difference is significant in all cases. Showing that KQCSA consistently obtain better values than the rest of the algorithms.

Finally, in Table 8, the results of the data set *mkp*.30, 500 are shown. For this dataset, QPSO* was included in the comparison in addition to BGWO, [55], and BFOA, [56], which corresponds to a binary fruit fly algorithm. In this last group of problems, when observing the best value indicator, we see that QPSO* obtained 15 best values and KQCSA, 17 best values. In the average of the best values, KQCSA was slightly higher, however, the Wilcoxon test indicates that the difference is not statistically significant. When analyzing the average indicator, we observed that QPSO* was higher in 23 instances and KQCSA in 7. In this case, the Wilcoxon test indicates that there is a significant difference in favor of QPSO*.

Additionally, we have incorporated Tables 9 and 10, in order to develop a better understanding of the comparisons. In the Average Gap column of Table 9, the value corresponds to the average of the Gaps calculated for each instance. Subsequently, in the Gap ratio column, the gap ratio of each algorithm is calculated using KQCSA as a basis for comparison. On the other hand, Table 10 identifies the main operators used by the compared algorithms, such as binarization methods, heuristics used in the repair, and local search operators.

We noted the following patterns:

- KQCSA shows to be a very efficient algorithm. KQCSA obtains top higher average solutions in two of the three sets of problems. No other gets similar results.

Table 6
OR-Library benchmarks MKP *mkp.5.500*.

		Best				| Average					
Instance	Best Known	BAAA	3R-BPSO	QPSO*	KQCSA	BAAA	3R-BPSO	QPSO*	KQCSA	Std	time(s)
5.500.0	120148	120066	120141	120130	**120148**	120013.7	120101.5	120105.7	**120131.1**	5.56	134.6
5.500.1	117879	117702	**117864**	117844	**117864**	117560.5	117825.5	117834.3	**117853.8**	6.67	167.1
5.500.2	121131	120951	121129	**121131**	**121131**	120782.9	121103.5	121092	**121126.9**	6.28	135.6
5.500.3	120804	120572	**120804**	120752	**120804**	120340.6	120772.3	120740.3	**120794.4**	8.14	145.8
5.500.4	122319	122231	**122319**	**122319**	**122319**	122101.8	122310.3	122300.7	**122317.1**	2.64	168.1
5.500.5	122024	121957	**122024**	**122024**	**122024**	121741.8	121981.1	121981.7	**122013.1**	9.05	187.2
5.500.6	119127	119070	**119127**	119094	**119127**	118913.4	119090.5	119075	**119122.4**	4.58	179.5
5.500.7	120568	120472	120545	120536	**120568**	120331.2	120534.7	120513.3	**120563.5**	8.32	182.3
5.500.8	121586	121052	121575	**121586**	121573	120683.6	121537.1	121527.3	**121557.0**	7.89	165.1
5.500.9	120717	120499	**120717**	120685	**120717**	120296.3	120674.8	120662.3	**120710.1**	10.03	176.5
5.500.10	218428	218185	218415	**218428**	218425	217984.7	218397.1	218394.7	**218414.3**	5.7	153.2
5.500.11	221202	220852	221191	**221202**	**221202**	220527.5	221167.4	221152.3	**221178.2**	7.72	146.7
5.500.12	217542	217258	217534	217528	2**17536**	217056.7	217518.3	217513	**217534.0**	1.09	158.1
5.500.13	223560	223510	**223560**	**223560**	**223560**	223450.9	223536.4	223537.7	**223559.5**	0	142.3
5.500.14	218966	218811	**218966**	218965	**218966**	218634.3	218933.6	218964.3	**218965.7**	0.81	135.8
5.500.15	220530	220429	**220530**	220527	**220530**	220375.9	220493.2	220498.7	**220528.3**	3.71	142.9
5.500.16	219989	219785	219987	219943	**219989**	219619.3	219973.2	219931.3	**219988.2**	0.99	138.1
5.500.17	218215	218032	218194	**218215**	**218215**	217813.2	218168.5	218185	**218198.1**	7.12	147.5
5.500.18	216976	216940	**216976**	**216976**	**216976**	216862	216942.4	216955.3	**216975.5**	0	144.3
5.500.19	219719	219602	219709	**219719**	**219719**	219435.1	219691.8	219698	**219717.1**	0.37	146.2
5.500.20	295828	295652	295805	**295828**	**295828**	295505	295786.7	295797.7	**295828.1**	0	149.4
5.500.21	308086	307783	308081	**308086**	**308086**	307577.5	308069.9	308064	**308079.6**	2.94	138.7
5.500.22	299796	299727	**299796**	299788	**299796**	299664.1	299761.6	299778	**299795.5**	0	157.3
5.500.23	306480	306469	306478	306480	306480	306385.0	306466.4	306466.3	**306477.8**	2.02	152.1
5.500.24	300342	300240	**300342**	**300342**	**300342**	300136.7	300322.2	300310	**300342.0**	0	144.6
5.500.25	302571	302492	302560	302560	302571	302376.0	302546.3	302547	**302563.0**	4.27	153.6
5.500.26	301339	301272	301327	301322	**301329**	301158	301310.4	301317.3	**301329.0**	0.37	156.7
5.500.27	306454	306290	306438	306422	**306454**	306138.4	306422.2	306409	**306446.6**	5.93	142.7
5.500.28	302828	302769	**302828**	**302828**	**302828**	302690.1	302812	302808.7	**302824.4**	2.27	148.4
5.500.29	299910	299757	299904	**299910**	299904	299702.3	299888.5	299885.3	**299902.9**	3.31	140.1
Average	214168.80	214014.23	214162.20	214157.67	**214167.03**	213861.95	214137.98	214134.9	**214160.39**		152.7
Wilcoxon *p*-value		1.62e−6	7.8 e-4	3.8e−3		1.88e−7	1.67 e-6	1.66e−6			

Table 7
OR-Library benchmarks MKP *mkp.10.500*.

	Best						Average					
Instance	Best Known	F&F	3R-BPSO	QPSO*	TE-DBS	KQCSA	3R-BPSO	QPSO*	TE-DBS	KQCSA	KQCSA (std)	time(s)
10.500.0	117821	117734	**117790**	117744	117811	117779	117699.3	117733.5	**117801.2**	117753.3	17.4	301.4
10.500.1	119249	119181	119155	119177	119249	**119188**	119125.0	119148.5	119223.0	**119178.8**	7.4	267.8
10.500.2	119215	119194	119211	**119215**	119215	**119215**	119094.9	119146.5	117625.6	**119165.1**	16.6	241.6
10.500.3	118829	118784	**118813**	118775	**118813**	118802	118754.2	118747.5	117625.8	**118776.6**	5.4	286.1
10.500.4	116530	116471	116470	116502	**116509**	**116509**	116402.8	116449.5	114312.4	**116462.1**	30.5	204.5
10.500.5	119504	119442	119461	119402	**119504**	119456	119426.6	119391.5	112503.7	**119434.2**	13.7	298.3
10.500.6	119827	119764	119764	**119827**	**119827**	119813	119720.5	119784.0	115629.1	**119778.3**	13.3	236.5
10.500.7	118344	118309	118288	118309	118301	**118320**	118243.3	**118282.5**	115531.9	118265.2	18.1	288.6
10.500.8	117815	117781	117752	117721	**117815**	117781	117698.3	117710.0	114204.0	**117762.1**	18.4	275.3
10.500.9	119251	119183	119186	**119251**	119231	119203	119127.3	**119200.5**	113622.8	119165.5	16.9	256.8
10.500.10	217377	217318	217345	217308	217377	**217366**	217283.0	217289.5	208710.2	**217327.8**	14.3	188.1
10.500.11	219077	219036	219053	**219077**	**219077**	219042	219002.1	219049.5	217277.2	**219026.2**	7.6	236.7
10.500.12	217847	**217797**	217755	**217797**	217377	217792	217743.8	**217772.0**	210172.3	217756.2	15.6	245.3
10.500.13	216868	216843	216832	**216868**	**216868**	216840	216770.9	**216826.0**	206178.6	216820.7	14.4	259.8
10.500.14	213873	213811	**213843**	213795	207017	**213843**	213777.9	213783.0	206656.0	**213811.6**	19.8	275.1
10.500.15	215086	215021	215058	**215086**	**215086**	215044	214992.9	**215053.5**	203989.5	215031.9	10.1	253.7
10.500.16	217940	217880	217896	217868	**217940**	217931	217858.3	217853.0	204828.9	**217885.2**	15.8	261.2
10.500.17	219990	219969	219949	219949	**219984**	**219984**	219906.6	219919.5	207881.6	**219966.5**	16.6	269.1
10.500.18	214382	214346	214351	**214382**	210735	214352	214286.0	**214364.0**	209787.6	214336.8	18.7	264.7
10.500.19	220899	220849	220840	220827	**220899**	220865	220806.3	220814.5	204435.7	**220848.7**	14.4	201.7
10.500.20	304387	304344	304344	304344	**304387**	304363	304311.8	304329.5	302658.8	**304355.1**	9.5	251.1
10.500.21	302379	302345	**302379**	302341	**302379**	302358	302315.6	302341.0	301658.6	**302344.5**	12.6	214.5
10.500.22	302417	302408	302396	**302417**	302416	302408	302348.8	302386.5	290859.9	**302399.6**	5.5	249.6
10.500.23	300784	300743	300743	**300784**	291295	**300784**	300712.2	**300763.5**	290021.4	300747.8	12.6	235.3
10.500.24	304374	304357	**304374**	304340	**304374**	304366	304341.0	304328.5	288950.1	**304354.8**	7.6	295.1
10.500.25	301836	301742	301796	**301836**	**301836**	301766	301712.7	**301787.5**	292061.8	301753.6	5.4	209.7
10.500.26	304952	304911	304949	**304952**	291446	304949	304944.0	304924.5	290516.2	**304949.0**	0.0	261.2
10.500.27	296478	296447	296438	296437	295342	**296457**	296416.7	296432.0	293125.5	**296444.2**	6.5	243.1
10.500.28	301359	301331	301353	301293	288907	**301357**	301290.4	301284.0	285293.4	**301327.4**	12.8	258.4
10.500.29	307089	**307078**	307072	307002	295358	307072	307004.9	306963.5	289552.4	**307071.0**	5.2	228.5
Average	212859.30	212813.97	212821.87	212820.87	210879.17	**212833.50**	212770.60	212795.30	205423.17	**212809.99**	12.75	251.5
Wilcoxon *p*-value		8.04e−6	8.2 e-3	0.031	0.048		1.63e−6	0.01	2.35e−6			

Table 8
OR-Library benchmarks MKP *mkp.30.500*.

	Best					Average					
Instance	Best Known	BGWO	QPSO*	BFAO	KQCSA	BGWO	QPSO*	BFAO	KQCSA	std	time(s)
30.500.0	116056	115814	**115991**	115759	115952	115104.3	**115906.0**	114779.4	115897.2	84.55	499.3
30.500.1	114810	114701	114684	114418	**114769**	114006.3	**114661.0**	113535.6	114615.0	66.48	602.4
30.500.2	116741	116531	**116712**	116329	116633	115807.1	**116642.5**	115375.1	116557.8	59.39	542.1
30.500.3	115354	115125	**115354**	115011	115236	114408.1	115062.5	114027.4	**115186.1**	30.59	567.1
30.500.4	116525	116312	**116435**	116098	116372	115662.7	**116378.5**	115231.6	116291.3	43.82	508.4
30.500.5	115741	115586	115594	115329	**115648**	114861.4	115583.5	114421.6	**115603.9**	40.19	631.2
30.500.6	114181	113939	**113987**	113930	113961	113301.8	**113936.5**	112890.8	113915.3	23.15	587.9
30.500.7	114348	114118	**114184**	113987	114141	113556.4	**114135.5**	113224.7	114010.0	58.53	605.1
30.500.8	115419	115153	**115419**	115023	115319	114564.9	**115271.0**	114126.3	115242.5	60.05	521.2
30.500.9	117116	116939	116909	116647	**116946**	116237.6	**116909.0**	115745.7	116895.8	42.42	568.2
30.500.10	218104	217938	**218068**	217803	218043	217340.6	**218068.0**	216904.4	217992.7	53.99	534.6
30.500.11	214648	214506	**214626**	214333	214512	213961.1	**214546.5**	213660.6	214461.2	40.99	487.1
30.500.12	215978	215817	215839	215662	**215883**	215265.3	**215839.0**	214984.5	215827.7	24.77	567.6
30.500.13	217910	217746	**217816**	217623	217787	217234.5	**217816.0**	216864.0	217767.3	17.05	582.5
30.500.14	215689	215495	215544	215248	**215596**	214998.8	**215544.0**	214589.0	215483.1	51.65	546.1
30.500.15	215919	215734	215753	215450	**215774**	215271.2	**215753.0**	214817.8	215695.4	43.32	577.5
30.500.16	215907	215761	215789	215673	**215871**	215194.5	**215784.5**	214913.8	215781.8	52.30	609.2
30.500.17	216542	216379	216387	216197	**216452**	215827.4	216387.0	215416.0	**216393.2**	44.85	634.1
30.500.18	217340	217251	217217	217170	**217290**	216622.8	217211.0	216340.2	**217229.1**	30.11	538.5
30.500.19	214739	214597	**214739**	214443	214681	214008.9	**214686.5**	213643.8	214605.2	56.97	582.9
30.500.20	301675	301627	**301643**	301440	**301643**	301132.0	**301635.0**	300800.1	301618.9	33.00	558.9
30.500.21	300055	299987	299965	299808	**299969**	299454.9	**299963.5**	299124.8	299929.7	22.88	564.1
30.500.22	305087	305028	305038	304859	**305055**	304476.8	**305038.0**	304141.2	304970.7	44.47	498.1
30.500.23	302032	301897	**301982**	301797	301935	301367.5	**301982.0**	301095.7	301924.5	19.86	486.4
30.500.24	304462	304411	304346	304146	**304397**	303853.1	304346.0	303518.2	**304387.0**	21.99	653.1
30.500.25	297012	296883	296892	296893	**296920**	296447.7	**296892.0**	296061.6	296841.6	21.37	706.2
30.500.26	303364	303232	**303287**	303131	**303287**	302787.6	**303287.0**	302484.2	303244.2	35.91	546.3
30.500.27	307007	306892	306915	306817	**306929**	306423.7	306915.0	306178.1	**306916.0**	23.93	605.8
30.500.28	303199	303077	**303169**	302950	303151	302592.6	**303169.0**	302319.7	303089.7	29.39	612.5
30.500.29	300596	300493	300449	300424	**300596**	299940.8	300449.0	299640.2	**300502.5**	30.49	571.4
	211451.87	211298.97	211357.77	211146.60	**211358.27**	210723.75	**211326.60**	210361.88	211295.76	40.28	569.9
Wilcoxon *p*-value		3.87e−6	0.95	1.73e−6		1.65e−6	1.7 e-3	1.67e−7			

Table 9
GAP ratio respect to KQCSA.

Problems	Algorithm	Avg Gap	Gap ratio
5.500	BAAA	0.0017410	37.20
	3R-BPSO	0.0001767	3.77
	QPSO	0.0002028	4.33
	KQCSA	0.0000468	1.00
10.500	3R-BPSO	0.0005195	1.76
	QPSO	0.0003731	1.26
	TE-DBS	0.0070486	23.82
	KQCSA	0.0002959	1.00
30.500	BGWO	0.0042801	4.48
	QPSO	0.0007865	0.82
	BFAO	0.0063735	6.68
	KQCSA	0.0009543	1.00

- KQCSA performs competitively in all complexity instances, however, for more complex solution spaces, the inclusion of an aggregating operator as KNN may induce excessive local trapping than stochastic search. This is in agreement with the fact that QPSO* displaces KQCSA in the third and most complex problem set with 30 restrictions and 500 elements.
- KNN-Quantum operator shown to be very effective in small and medium search spaces. Indeed, KQCSA obtains top average and numbers of best solutions for instances of 5 and 10 restrictions.
- The Quantum PSO operator appears to scan the most complex spaces more effectively. We see that in the more complex solution space of 30 constraints and 500 items, QPSO* outperforms second-ranked KQCSA by around 20% on the relative index.
- When comparing KQCSA with the methods that use TF, which corresponds to a mechanism that does not alter the movements of the metaheuristics directly, regardless of the size of the problem and the number of constraints and operators involved, KQCSA outperforms the other methods.

In summary, when analyzing the results of the comparisons made in the Tables 6, 7, and 8. KQCSA performs very well on the *mkp*5.500 and *mkp*10.500 problems where the algorithm was always superior to the others. However, for problem *mpk*30.500 on average QPSO* performed better than KQCSA. This opens the opportunity to explore improvements in the algorithm, also considering that the standard deviations and the average convergence times increase significantly with respect to the results obtained in the *mkp*5.500 and *mkp*10.500 instances.

6. Conclusions

In this article, we have proposed a hybrid KNN quantum cuckoo search algorithm. This hybrid algorithm incorporates the concept of k-nearest neighbor in the movement of the solutions, as well as in updating the solutions. First of all, a brief state-of-the-art of the hybrid techniques was made, highlighting the contribution of hybridization in the results. Later, the proposed hybrid algorithm was used to solve medium and large instances of the multidimensional knapsack problem. The contribution of the KNN concept in the movement of solutions was studied, as well as in the method of updating them. To achieve this objective, two operators QCSA and SUQCSA were designed. Additionally, the iteration times involved in fit the k-nearest neighbors and subsequent execution in obtaining the neighbors were estimated. Besides, the average times used to solve each instance are also reported. Finally, when we compare our proposal with different algorithms of the state-of-the-art, we observe that most of the time our algorithm is capable of improving the previous results.

Table 10
Summary of algorithm operators.

Complexity Scale	Algorithm Rel Order	Principal Operators
5 restriction, 500 elements	KQCSA	KNN-Q-operator/KNN update/RMRO
	3R-BPSO	TF/Local Search/Three ratio
	QPSO*	Q-operator/Local search/Pseudo-utility ratio
	BAAA	TF/Local Search/RMRO
10 restrictions, 500 elements	KQCSA	KNN-Q-operator/KNN update/RMRO
	QPSO*	Q-operator/Local search/Pseudo-utility ratio
	3R-BPSO	TF/Local Search/Three ratio
	TE-DBS	TF/Local Search/RMRO
30 restrictions, 500 elements	QPSO*	Q-operator/Local search/Pseudo-utility ratio
	KQCSA	KNN-Q-operator/KNN update/Pseudo-utility ratio
	BGWO[a]	BGWO/elite population strategy/Pseudo-utility ratio
	BFOA[a]	TF/Pseudo-utility ratio/rank of the item

Relative mean resource occupation (RMRO); Transfer Function (TF); K-Nearest Neighborhood (KNN); Quantum operator (Q-operator).
[a]In the case of BGWO, this algorithm corresponds to a specific binarization method in which the equations of the original metaheuristic is modified.

As future lines of research, we plan to apply our algorithm to other combinatorial problems. At first, we would like to evaluate related problems such as the multiple-choice multidimensional knapsack problem or the set union knapsack problem, to later extend it to other combinatorial problems. Another line of research that we would like to explore is the use of dynamic parameters. In Section 5.1, fixed parameters were defined. We intuit that these can be modified in the execution of the algorithm and, as a consequence of this, obtain more robust algorithms. Here we plan to incorporate reinforcement learning techniques so that they help us in modifying the parameters. Finally, it would be interesting to adapt the concept of k-nearest neighbors used in KQCSA to other quantum metaheuristics, such as QPSO*. Then apply the designed algorithm to the MKP or to some related problem.

CRediT authorship contribution statement

José García: Conceptualization, Methodology, Visualization, Investigation, Writing - original draft, Writing - review & editing. **Carlos Maureira:** Conceptualization, Methodology, Data curation, Writing - original draft, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

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

References

[1] S. Korkmaz, A. Babalik, M. Kiran, An artificial algae algorithm for solving binary optimization problems, Int. J. Mach. Learn. Cybern. 9 (2018) 1233–1247.
[2] V. Penadés-plà, T. García-segura, V. Yepes, Robust design optimization for low-cost concrete box-girder bridge, Mathematics 8 (2020) 398.
[3] H. Guo, B. Liu, D. Cai, T. Lu, Predicting protein–protein interaction sites using modified support vector machine, Int. J. Mach. Learn. Cybern. 9 (2018) 393–398.
[4] N. Al-madi, H. Faris, S. Mirjalili, Binary multi-verse optimization algorithm for global optimization and discrete problems, Int. J. Mach. Learn. Cybern. 10 (2019) 3445–3465.
[5] E. Talbi, Combining metaheuristics with mathematical programming, constraint programming and machine learning, Ann. Oper. Res. 240 (2016) 171–215.
[6] Y. Tsao, T. Vu, L. Liao, Hybrid heuristics for the cut ordering planning problem in apparel industry, Comput. Ind. Eng. (2020) 106478.
[7] M. Caserta, S. Voß, Metaheuristics: Intelligent Problem Solving, Springe, 2009.
[8] D. Schermer, M. Moeini, O. Wendt, A matheuristic for the vehicle routing problem with drones and its variants, Transp. Res. C 106 (2019) 166–204.
[9] A. Juan, J. Faulin, S. Grasman, M. Rabe, G. Figueira, A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems, Oper. Res. Perspect. 2 (2015) 62–72.
[10] J. Panadero, J. Doering, R. Kizys, A. Juan, A. Fito, A variable neighborhood search simheuristic for project portfolio selection under uncertainty, J. Heuristics 26 (2020) 353–375.
[11] V. Yepes, J. Martí, J. García, Black hole algorithm for sustainable design of counterfort retaining walls, Sustainability 12 (2020) 2767.
[12] J. Doering, R. Kizys, A. Juan, Fitó, O. Polat, Metaheuristics for rich portfolio optimisation and risk management: Current state and future trends, Oper. Res. Perspect. 6 (2019) 100121.
[13] X. Yu, Y. Zhou, X. Liu, The two-echelon multi-objective location routing problem inspired by realistic waste collection applications: The composable model and a metaheuristic algorithm, Appl. Soft Comput. (2020) 106477.
[14] J. García, J. Martí, V. Yepes, The buttressed walls problem: An application of a hybrid clustering particle swarm optimization algorithm, Mathematics 8 (2020) 862.
[15] G. Sayed, A. Tharwat, A. Hassanien, Chaotic dragonfly algorithm: An improved metaheuristic algorithm for feature selection, Appl. Intell. 49 (2019) 188–205.
[16] L. Calvet, J. Dearmas, D. Masip, A. Juan, Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs, Open Math. 15 (2017) 261–280.
[17] E. Talbi, Machine learning into metaheuristics: A survey and taxonomy of data-driven metaheuristics.
[18] M. López-ibáñez, J. Dubois-lacoste, L. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Oper. Res. Perspect. 3 (2016) 43–58.
[19] J. Ries, P. Beullens, A semi-automated design of instance-based fuzzy parameter tuning for metaheuristics based on decision tree induction, J. Oper. Res. Soc. 66 (2015) 782–793.
[20] A. Shahzad, N. Mebarki, Data mining based job dispatching using hybrid simulation–optimization approach for shop scheduling problem, Eng. Appl. Artif. Intell. 25 (2012) 1173–1181.
[21] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, A. Kyek, Optimization of global production scheduling with deep reinforcement learning, Procedia CIRP 72 (2018) 1264–1269.
[22] M. Jiang, Z. Huang, L. Qiu, W. Huang, G. Yen, Transfer learning-based dynamic multiobjective optimization algorithms, IEEE Trans. Evol. Comput. 22 (2017) 501–514.
[23] J. García, B. Crawford, R. Soto, G. Astorga, A clustering algorithm applied to the binarization of swarm intelligence continuous metaheuristics, Swarm Evol. Comput. 44 (2019) 646–664.
[24] L. Saviniec, M. Santos, A. Costa, L. Dossantos, Pattern-based models and a cooperative parallel metaheuristic for high school timetabling problems, Eur. J. Oper. Res. 280 (2020) 1064–1081.
[25] A. Deleón, E. Lalla-ruiz, B. Melián-batista, J. Moreno-vega, A machine learning-based system for berth scheduling at bulk terminals, Expert Syst. Appl. 87 (2017) 170–182.

[26] A. Gutierrez-rodríguez, S. Conant-pablos, J. Ortiz-bayliss, H. Terashima-marín, Selecting meta-heuristics for solving vehicle routing problems with time windows via meta-learning, Expert Syst. Appl. 118 (2019) 470–481.
[27] J. Toutouh, E. Alba, Parallel multi-objective metaheuristics for smart communications in vehicular networks, Soft Comput. 21 (2017) 1949–1961.
[28] S. Martin, D. Ouelhadj, P. Beullens, E. Ozcan, A. Juan, E. Burke, A multi-agent based cooperative approach to scheduling and routing, Eur. J. Oper. Res. 254 (2016) 169–178.
[29] R. Tyasnurita, E. özcan, A. Shahriar, R. John, Improving performance of a hyper-heuristic using a multilayer perceptron for vehicle routing.
[30] S. Choong, L. Wong, C. Lim, Automatic design of hyper-heuristic based on reinforcement learning, Inform. Sci. 436 (2018) 89–107.
[31] F. Ozsoydan, M. özdemir, Iterated greedy algorithms enhanced by hyper-heuristic based learning for hybrid flexible flowshop scheduling problem with sequence dependent setup times: A case study at a manufacturing plant, Comput. Oper. Res. (2020) 105044.
[32] V. Penadés-plà, T. García-segura, V. Yepes, Accelerated optimization method for low-embodied energy concrete box-girder bridge design, Eng. Struct. 179 (2019) 556–565.
[33] M. Esfe, P. Razi, M. Hajmohammad, S. Rostamian, W. Sarsam, A. Arani, M. Dahari, Optimization, modeling and accurate prediction of thermal conductivity and dynamic viscosity of stabilized ethylene glycol and water mixture Al2o3 nanofluids by NSGA-II using ANN, Int. Commun. Heat Mass Transfer 82 (2017) 154–160.
[34] P. Satrio, T. Mahlia, N. Giannetti, K. Saito, Optimization of HVAC system energy consumption in a building using artificial neural network and multi-objective genetic algorithm, Sustain. Energy Technol. Assess. 35 (2019) 48–57.
[35] S. He, W. Chen, X. Mu, W. Cui, Constrained optimization model of the volume of initial rainwater storage tank based on ANN and PSO, Environ. Sci. Pollut. Res. (2020) 1–14.
[36] T. Rawlins, A. Lewis, T. Kipouros, Repairing blackbox constraint violations in multi-objective optimisation by use of decision trees.
[37] N. Veček, M. Mernik, B. Filipič, M. črepinšek, Parameter tuning with chess rating system (CRS-Tuning) for meta-heuristic algorithms, Inform. Sci. 372 (2016) 446–469.
[38] G. Shankar, V. Mukherjee, Quasi oppositional harmony search algorithm based controller tuning for load frequency control of multi-source multi-area power system, Int. J. Electr. Power Energy Syst. 75 (2016) 289–302.
[39] A. Fréville, The multidimensional 0–1 knapsack problem: An overview, Eur. J. Oper. Res. 155 (2004) 1–21.
[40] M. Garey, D. Johnson, NP-completeness. Computers and intractability, p. 197.
[41] A. Tzanetos, G. Dounias, Nature inspired optimization algorithms or simply variations of metaheuristics? Artif. Intell. Rev. (2020) 1–22.
[42] X. Yang, Nature-inspired optimization algorithms: challenges and open problems, J. Comput. Sci. (2020) 101104.
[43] M. Vasquez, J. Hao, A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite, Comput. Optim. Appl. 20 (2001) 137–157.
[44] M. Yang, An efficient algorithm to allocate shelf space, Eur. J. Oper. Res. 131 (2001) 107–118.
[45] H. Pirkul, An integer programming model for the allocation of databases in a distributed computer system, Eur. J. Oper. Res. 26 (1986) 401–411.
[46] P. Gilmore, R. Gomory, The theory and computation of knapsack functions, Oper. Res. 14 (1966) 1045–1074.
[47] Y. Vimont, S. Boussier, M. Vasquez, Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem, J. Comb. Optim. 15 (2008) 165–178.
[48] S. Boussier, M. Vasquez, Y. Vimont, S. Hanafi, P. Michelon, A multi-level search strategy for the 0–1 multidimensional knapsack problem, Discrete Appl. Math. 158 (2010) 97–109.
[49] R. Mansini, M. Speranza, Coral: An exact algorithm for the multidimensional knapsack problem, Informs J. Comput. 24 (2012) 399–415.
[50] S. Hanafi, A. Freville, An efficient tabu search approach for the 0–1 multidimensional knapsack problem, Eur. J. Oper. Res. 106 (1998) 659–675.
[51] M. Khemakhem, B. Haddar, K. Chebil, S. Hanafi, A filter-and-fan metaheuristic for the 0-1 multidimensional knapsack problem, Int. J. Appl. Metaheuristic Comput. 3 (2012) 43–63.
[52] B. Haddar, M. Khemakhem, S. Hanafi, C. Wilbaut, A hybrid quantum particle swarm optimization for the multidimensional knapsack problem, Eng. Appl. Artif. Intell. 55 (2016) 1–13.
[53] M. Chih, Three pseudo-utility ratio-inspired particle swarm optimization with local search for multidimensional knapsack problem, Swarm Evol. Comput. 39 (2018) 279–296.
[54] J. García, E. Lalla-ruiz, S. Voß, E. Droguett, Enhancing a machine learning binarization framework by perturbation operators: Analysis on the multidimensional knapsack problem, Int. J. Mach. Learn. Cybern. (2020) 1–20.
[55] K. Luo, Q. Zhao, A binary grey wolf optimizer for the multidimensional knapsack problem, Appl. Soft Comput. 83 (2019) 105645.
[56] L. Wang, X. Zheng, S. Wang, A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem, Knowl.-Based Syst. 48 (2013) 17–23.
[57] X. Zhang, C. Wu, J. Li, X. Wang, Z. Yang, J. Lee, K. Jung, Binary artificial algae algorithm for multidimensional knapsack problems, Appl. Soft Comput. 43 (2016) 583–595.
[58] J. Liu, C. Wu, J. Cao, X. Wang, K. Teo, A binary differential search algorithm for the 0–1 multidimensional knapsack problem, Appl. Math. Model. 40 (2016) 9788–9805.
[59] J. Zhao, S. Liu, M. Zhou, X. Guo, L. Qi, An improved binary cuckoo search algorithm for solving unit commitment problems: Methodological description, IEEE Access 6 (2018) 43535–43545.
[60] J. García, F. Altimiras, A. Peña, G. Astorga, O. Peredo, A binary cuckoo search big data algorithm applied to large-scale crew scheduling problems, Complexity 2018 (2018).
[61] M. Dalali, H. Karegar, Optimal PMU placement for full observability of the power network with maximum redundancy using modified binary cuckoo optimisation algorithm, Iet Gener. Transm. Distrib. 10 (2016) 2817–2824.
[62] J. García, V. Yepes, J. Martí, A hybrid k-means cuckoo search algorithm applied to the counterfort retaining walls problem, Mathematics 8 (2020) 555.
[63] T. Hey, Quantum computing: An introduction, Comput. Control Eng. J. 10 (1999) 105–112.
[64] X. Kong, L. Gao, H. Ouyang, S. Li, Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm, Comput. Oper. Res. 63 (2015) 7–22.
[65] F. Wilcoxon, Individual Comparisons by Ranking Methods, Springe, 1992.
[66] W. Hays, R. Winkler, Statistics: Probability, inference, and decision.
[67] J. García, B. Crawford, R. Soto, C. Castro, F. Paredes, A k-means binarization framework applied to multidimensional knapsack problem, Appl. Intell. 48 (2018) 357–380.
[68] B. Crawford, R. Soto, G. Astorga, J. García, C. Castro, F. Paredes, Putting continuous metaheuristics to work in binary search spaces, Complexity 2017 (2017).