Appl Intell (2018) 48:357–380
DOI 10.1007/s10489-017-0972-6

A k-means binarization framework applied to multidimensional knapsack problem

José García[1,2] · Broderick Crawford[2] · Ricardo Soto[2] · Carlos Castro[3] · Fernando Paredes[4]

Published online: 11 July 2017

Abstract The multidimensional knapsack problem (MKP) is one of the widely known integer programming problems. The MKP has received significant attention from the operational research community for its large number of applications. Solving this NP-hard problem remains a very interesting challenge, especially when the number of constraints increases. In this paper we present a k-means transition ranking (KMTR) framework to solve the MKP. This framework has the property to binarize continuous population-based metaheuristics using a data mining k-means technique. In particular we binarize a Cuckoo Search and Black Hole metaheuristics. These techniques were chosen by the difference between their iteration mechanisms. We provide necessary experiments to investigate the role of key ingredients of the framework. Finally to demonstrate the efficiency of our proposal, MKP benchmark instances of the literature show that KMTR competes with the state-of-the-art algorithms.

Keywords Metaheuristics · Multidimensional knapsack problem · Binarization · Data mining · k-means

✉ José García
joseantonio.garcia@telefonica.com

Broderick Crawford
broderick.crawford@pucv.cl

Ricardo Soto
ricardo.soto@pucv.cl

1 Telefónica I+D, Av. Manuel Montt 1404, Third Floor, Providencia, Santiago, Chile

2 Pontificia Universidad Católica de Valparaíso, 2362807 Valparaíso, Chile

3 Universidad Técnica Federico Santa María, Valparaíso, Chile

4 Escuela de Ingeniería Industrial, Universidad Diego Portales, Santiago, Chile

1 Introduction

The knapsack problem has multiple applications in science and engineering. For example capital budgeting and project selection applications [47, 54, 71]. The MKP has also been introduced to model problems like cutting stock [25], loading problems [62], allocation of processors in a distributed data processing [22], delivery in vehicles with multiple compartments [10] and self-sufficiency problems [64].

Numerous methods have been developed to solve the MKP. The exact methods were applied in the 80's to solve MKP [5, 20, 46]. They generate a variety of methods including dynamic programming, branch-and-bound, network approach and reduction schemes. The exact methods have made possible the solution of middle size MKP instances. The major drawback of these methods remains the temporal complexity when dealing with large instances. Therefore, many researchers focus on heuristic and metaheuristic search methods which can produce solutions of good qualities in a reasonable amount of time. In recent years, many bio-inspired methods, such as Genetic algorithms [45], Particle Swarm Optimization (PSO) [6, 14] Firefly algorithm [7], Ant Colony Optimization [12], and a binary Fruitfly [70] have been proposed to solve large instances of the MKP.

Many of these bio-inspired methods, are working in continuous spaces and they have had to be adapted to a binary version. Examples of these adaptations are found in

Harmony Search (HS) [23], Swarm Intelligence [9], Wind driven optimization [83],Differential Evolution Algorithm (DE) [26, 82], PSO [33], Magnetic Optimization Algorithm (MOA) [68], and Gravitational Search Algorithm (GSA) [58], Swarm Intelligence [53], and Black Hole [21].

In many of these adaptations, the transfer functions are used as a general mechanism to perform binarization. Examples of using transfer functions are found in the firefly algorithm [50, 57], PSO [33, 35], Artificial Bee Colony [16], Cuckoo search [65], Teach learning [1]. In [80] a binary artificial algae algorithm (BAAA) solves medium and large MKP with very good results. This algorithm in addition to transfer function, used an interesting heuristics for solution repair, and elitist local search to improve the solutions. In [43] the authors propose a binary differencial search (BDS) algorithm also with good results to solve the knapsack problem based in Brownian motion and a v-shape transfer function. A hybrid harmony search-based algorithm (HHS) [78] obtained interesting results in problems of medium size.

In this paper, a general framework is proposed to binarize continuous metaheuristics. This framework is called k-means transition ranking (KMTR) which is composed of three operators. The main operator corresponds to k-means transition operator. This operator performs the binarization process and is complemented with local search and perturbation operators. The main goal of this work corresponds to evaluate our framework when dealing with an NP-hard combinatorial optimization problem such as the MKP. To develop the evaluation, we used two metaheuristics: Cuckoo Search and Black Hole. These metaheuristics were chosen by the difference between their iteration mechanisms. Cuckoo uses iteration through Lévy flights while Black Hole uses a simplified PSO mechanism. Additionally, it is interesting to use our framework in metaheuristics that have already solved the MKP as Cuckoo Search [24, 38] and others like Black Hole that to our knowledge have not been solved the MKP.

For appropriate evaluation of our framework, a method of estimating parameters is developed. Subsequently experiments were developed that shed light on the contribution of the different operators at the end result. Finally our framework was compared with recent algorithms that use transfer functions as binarization method. For this purpose we use different sets of tests problems from the OR-Library.[1] We compared our framework with the BAAA algorithm published by [80], and the algorithms TR-BDS and TE-BDS reported in [43], both algorithms are state of the art published in 2016. The numerical results show that KMTR achieves highly competitive results.

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

The remainder of this paper is organized as follows. Section 2 briefly introduces the Knapsack problem. In Section 3 other binarization works are presented. In Section 4 we explain the k-means transition ranking framework. The results of numerical experiment are presented in Section 5. Finally we provide the conclusions of our work.

2 KnapSack problem

The multidimensional knapsack problem [72] belongs to the class of NP-hard problems. MKP corresponds to a model of resource allocation, whose objective is to select a subset of objects that produce the greatest benefit considering certain capacity constraints. Each object j consumes a different amount of resources in each dimension. Also each object has a profit associated. Formally the MKP can be set as:

$$\text{maximize} \sum_{j=1}^{n} p_j x_j \tag{1}$$

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

$$\text{with } x_j \in \{0, 1\} \ , \ j \in \{1, \ldots, n\} \tag{3}$$

Where p_j is the profit for the item j, c_{ij} corresponds to the consumption of resources of item j in the dimension i, and b_i is the capacity constraint of each dimension i. The representation of a solution of the problem is modelled naturally in binary form where 0 in the j-th position means that the j item is not included in the Knapsack and 1 indicates that j is included.

3 Related work

There is a set of metaheuristic techniques that were designed to operate in continuous spaces. Examples of these techniques are Artificial Bee Colony [34], Particle Swarm Optimization [61], Black Hole [29], Cuckoo Search [75], Bat Algorithm [74], FireFly Algorithm [73], FruitFly [52], Artificial Fish Swarm [42], Gravitational Search Algorithm [58]. Moreover, in operational research, there are a lot of problems that are combinatorial and non-polynomial type [37]. So naturally, the idea arises of applying these continuous metaheuristics to combinatorial problems which are solved in discrete spaces. These adaptations are generally not trivial and have given rise at different lines of research.

When a review is made in the literature of binarization techniques, two main groups appear. A first group corresponds to general binarization frameworks. In these frameworks there is a mechanism that allows to transform any continuous metaheuristics in a binary one, without altering the metaheuristics operators. In this category the main frameworks used are: Transfer Functions and Angle Modulation. The second group corresponds to binarizations developed specifically for a metaheuristic. Within this second group we found techniques such as Quantum Binary and Set based approach.

Transfer functions The transfer function is the most used binarization method. It was introduced by [33]. The transfer function is a very cheap operator, his range provides probabilities values and tries to model the transition of the particle positions. This function is responsible for the first step of the binarization method which corresponds to map the $\mathbb{R}^n$ solutions in $[0, 1]^n$ solutions. Two types of functions have been used in the literature, the S-shaped [76], and V-shaped [17]. The Second Step is to apply a binarization rule to the result of the transfer function. Examples of binarization rules are complement, roulette, static probability, and elitist [17].

In [35], this framework was used to optimize sizing of Capacitor Banks in Radial Distribution Feeders. In [59], transfer functions were used for the analysis of bulk power systems. This approach has also been used to solve the set covering problem using Binary Firefly Algorithm [17]. Soto et al. [65] used Cuckoo Search Algorithm applied to the set covering problem. To solve the unit commitment problem Yang et al. in [76] used Firefly and PSO algorithms. The knapsack crystosystem was approached in [50]. Network and reliability constrained problems were solved in [11] and the Knapsack problem was solved by Zhang et al. [80] all using using Firefly algorithm.

Angle modulation This method uses the trigonometric function shown in (4). This function has four parameters which control the frequency and shift of the trigonometric function.

$$g_i(x_j) = \sin(2\pi(x_j - a_i)b_i \cos(2\pi(x_j - a_i)c_i)) + d_i \quad (4)$$

This method was first applied in PSO, using a set of benchmark functions. Let a binary problem of n-dimension, and $X = (x_1, x_2, ...x_n)$ a solution. We start with a four dimensional search space. Each dimension represents a coefficient of the (4). 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 rule (5) is applied:

$$b_{ij} = \begin{cases} 1 \text{ if } g_i(x_j) \geq 0 \\ 0 \text{ otherwise} \end{cases} \quad (5)$$

Then for each initial 4-dimension solution (a_i, b_i, c_i, d_i), we get a binary n-dimension solution $(b_{i1}, b_{i2}, ..., b_{in})$. This is a feasible solution of our n-binary problem. The Angle modulate technique has been applied to network reconfiguration problems [44] using a binary PSO method, to multi-user detection technique [66] using a binary adaptive evolution algorithm, and to the antenna position problem using a angle modulate binary bat algorithm [77].

Quantum binary approach In the line of research that involves the areas of Evolutionary Computing (EC) and Quantum Computing, there are three categories of algorithms [79]

1. Quantum evolutionary algorithms: These algorithms focus on the application of EC algorithms 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: These algorithms concentrate on the generation of new EC algorithms using some concepts and principles of Quantum Computing.

In particular the Quantum Binary Approach, belongs to Quantum-inspired evolutionary algorithms. In this sense these algorithms adapt the concepts of q-bits and superposition to work on normal computers.

In the quantum binary approach method, each feasible solution has a position $X = (x_1, x_2, .., x_n)$ and the quantum q-bits vector $Q = [Q_1, Q_2, ..., Q_n]$. Q represents the probability of x_j take 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 Q vector is specific to each metaheuristic.

The Quantum Swarm optimization algorithm has been applied to a combinatorial optimization in [69], cooperative approach in [81], knapsack problem in [63], and power quality monitor in [31]. The Quantum Differential Evolution algorithm was applied to the knapsack problem in [30], combinatorial problems [3], and image threshold methods in [18]. Using Cuckoo search metaheuristic a Quantum algorithm was applied to the knapsack problem [38], and bin packing problem [40]. A Quantum Ant Colony Optimization was applied to image threshold in [18]. Using Harmony Search in [39], and Monkey algorithm in [84], quantum binarizations were applied to the knapsack problem.

The general binarization frameworks have the difficulty of producing Spacial Disconnect [41]. The Spacial Disconnect, occurs when close solutions generated by metaheuristics in the continuous space, are not converted into close solutions in discrete space. Informally we can think in a loss of framework continuity. This phenomenon of Spacial Disconnect has the consequence that the properties of exploration and exploitation are altered and therefore the precision and convergence of the metaheuristic worsen. A study of how transfer functions affect exploration and exploitation properties was developed in [60]. For Angle Modulation the study was developed in [41].

On the other hand, specific binarization algorithms, which modify the operators of the metaheuristic, are susceptible to problems such as Hamming cliffs, loss of precision, search space discretization and the curse of dimension [41]. This was studied by Pampara in [51] and for the particular case of PSO by Chen in [13]. In the investigation of Chen, he observed that the parameters of the Binary PSO change the speed behavior of the original metaheuristic.

In this article, a k-means binarization framework is proposed which does not modify the original metaheuristic. The main operator of this framework, establishes a relation between the displacement of particles in the continuous space and the transition of probability in the discrete space. This relationship is established through the clustering of the displacements. To each group generated by clustering a transition probability is assigned. With this mechanism, it is expected that the exploration and exploitation properties will not be altered, and therefore to observe good results of convergence and precision of the binarized algorithms in the resolution of combinatorial problems.

4 k-means transition ranking framework

The Proposed KMTR framework has four main modules. The first module corresponds to the initialization of the feasible solutions (Section 4.1). Once the initialization of the particles is performed, it is consulted if the detention criterion is satisfied. This criterion includes a maximum of iterations. In the case that the optimal solution is known, this is also included as stopping criterion. Subsequently if the criterion is not satisfied, the transition ranking operator is executed (Section 4.2). This module is responsible for performing the iteration of solutions. Once the transitions of the different solutions are made, we compare the resulting solutions with the best solution previously obtained. In the event that a superior solution is found, this replaces the previous one. When a replacement occurs, the new solution is subjected to a local search operator. This operator corresponds to our third module (Section 4.4). Finally, having met a number of iterations where there has not been a replacement for the best solution, a perturbation operator is used (Section 4.5). The general algorithm scheme is detailed in Fig. 1.

4.1 Initialization and element weighting

KMTR framework uses a binarization of population-based metaheuristics to try to find the optimum. Each of these possible solutions, is generated as follows: First we select an item randomly. Subsequently we consulted the constraints

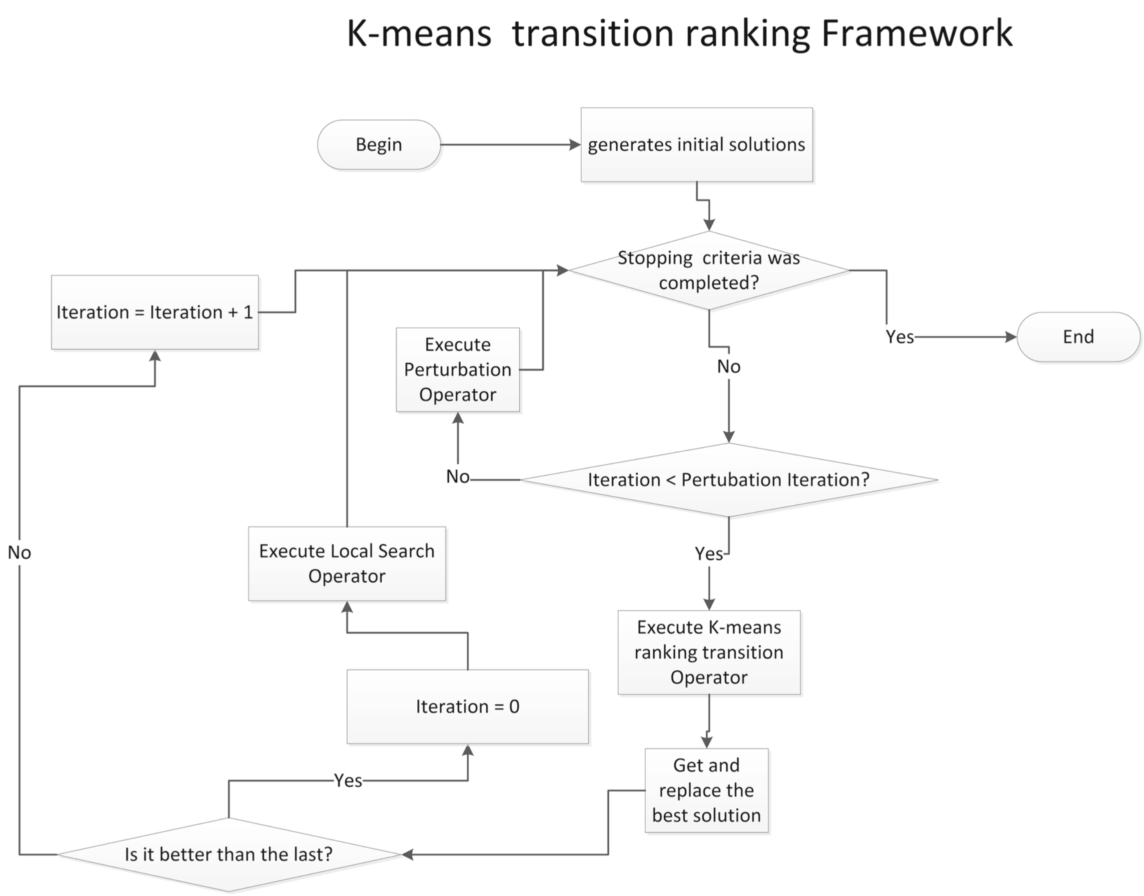


Fig. 1 Flowchart of general framework of k-means transition ranking algorithm

of our problem if there are other elements that can be incorporated. The list of possible elements to be incorporated is obtained, the weight for each of these elements is calculated and the best element is selected. The procedure continues until no more elements can be incorporated. The initialization algorithm is detailed in Fig. 2.

Several techniques were proposed in the literatures, to calculate the weight of each element. For example [55] introduced the pseudo-utility in the surrogate duality approach. The pseudo-utility of each variable was given in (6). The variable w_j is the surrogate multiplier between 0 and 1 which can be viewed as shadow prices of the j-th constraint in the linear programming(LP) relaxation of the original MKP

$$\delta_i = \frac{p_i}{\sum_{j=1}^{m} w_j c_{ij}} \tag{6}$$

Another more intuitive measure is proposed by [36]. This measure is focused on the average occupancy of resources. Its equation is shown in (7).

$$\delta_i = \frac{\sum_{j=1}^{m} \frac{c_{ij}}{m b_j}}{p_i} \tag{7}$$

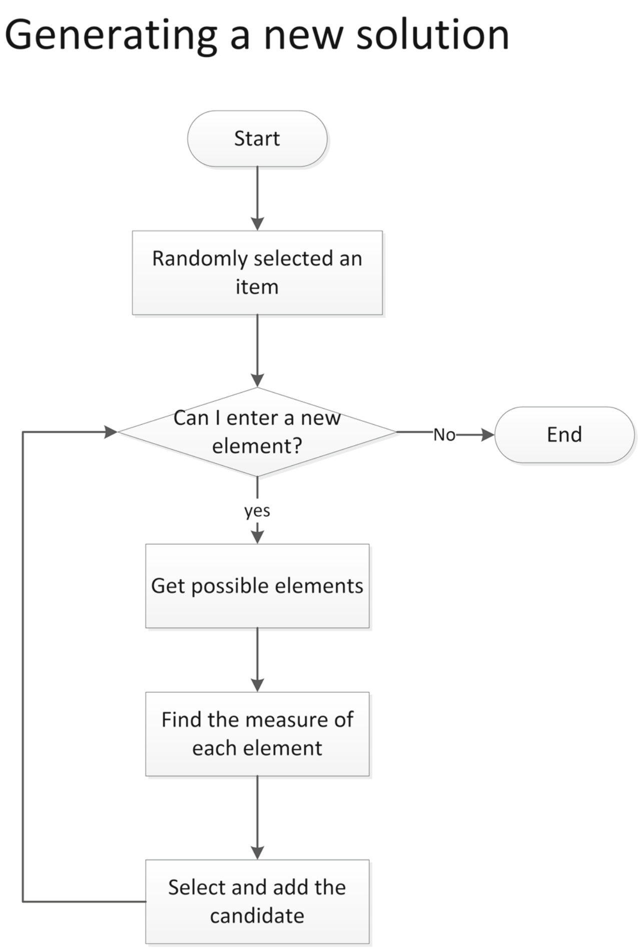


Fig. 2 Flowchart of generation of a new solution

In this paper, we propose a variation of this last measure focused on the average occupation. However this variation considers the elements that exist in backpacks to calculate the average occupancy. In each iteration depending on the selected items in the solution the measure is calculated again. The equation of this new measure is shown in (8).

$$\delta_i = \frac{\sum_{j=1}^{m} \frac{c_{ij}}{m(b_j - \sum_{i \in S} c_{ij})}}{p_i} \tag{8}$$

4.2 k-means transition ranking operator

Consider that our metaheuristic is continuous and population based. Due to its iterative nature, it needs to update the position of particles at each iteration. When the metaheuristic is continuous, this update is performed in $\mathbb{R}^n$ space. In (9), the update position is presented in a general manner. The x_{t+1} variable represents the x position of the particle at time t+1. This position is obtained from the position x at time t plus a Δ function calculated at time t+1. The function Δ is proper to each metaheuristic and produces values in $\mathbb{R}^n$. For example in Cuckoo Search $\Delta(x) = \alpha \oplus Levy(\lambda)(x)$, in Black Hole $\Delta(x) = \text{rand} \times (x_{bh}(t) - x(t))$ and in the Firefly, Bat and PSO algorithms Δ can be written in simplified form as $\Delta(x) = v(x)$.

$$x_{t+1} = x_t + \Delta_{t+1}(x(t)) \tag{9}$$

In the k-means transition ranking operator, a model for transitions in a discrete space is proposed. This model is based on considering the movements generated by the metaheuristic in each dimension for all particles. $\Delta^i(x)$ corresponds to the magnitude of the displacement $\Delta(x)$ in the i-th position. Subsequently these displacement are grouped using the magnitude of the displacement $\Delta^i(x)$. This grouping is done using the k-means technique where k represents the number of clusters used. Finally, a generic P_{tr} function given by the (10) is proposed to assign a transition probability.

$$P_{tr} : \mathbb{Z}/k\mathbb{Z} \rightarrow [0, 1] \tag{10}$$

A transition probability through the function P_{tr} is assigned to each group. Naturally, this P_{tr} function is modelled as a cumulative probability function. For the case of this study, we particularly use the linear function given in (11). In this equation, N(x^i) indicates the location of the group to which $\Delta^i(x)$ belongs. The α coefficient, corresponds to the transition probability and β to the transition separation coefficient. Both parameters are estimated in each metaheuristic. For our particular case, N(x^i) = 0 corresponds to elements belonging to the group that has the

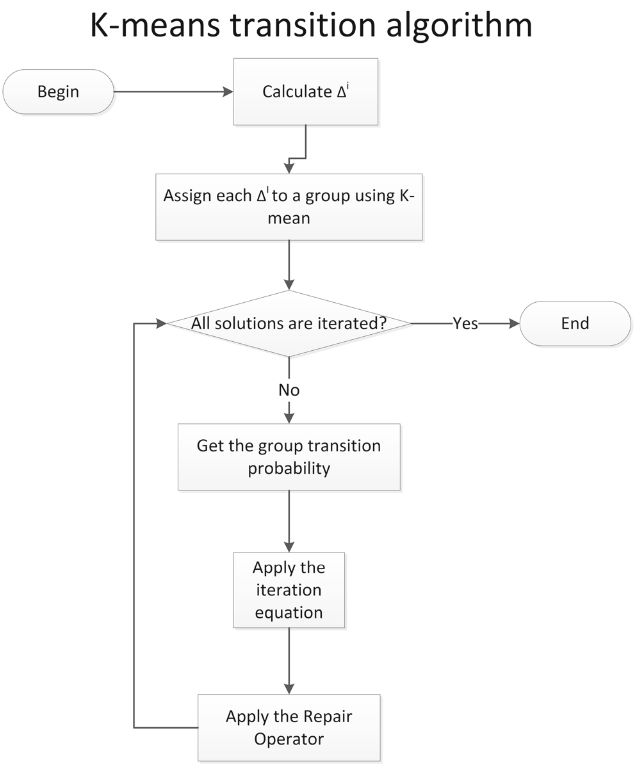


Fig. 3 Flowchart of transition ranking operator

lowest Δ^i values. N(x^i) $=$ 7 corresponds to the group of elements that have the greatest Δ^i values.

$$P_{tr}(x^i) = P_{tr}(N(x^i)) = \alpha + \beta N(x^i)\alpha \tag{11}$$

The algorithm flow chart is described in Fig. 3, and an illustration is shown in Fig. 4. The k-means transition ranking operator starts calculating Δ^i for each of the particles. This step is specific in each metaheuristic. Subsequently the particles are grouped using k-means clustering technique and the magnitude of Δ^i as distance. With the group assigned to each particle we obtain the probability of transition using (11). Afterwards the transition of each particle is performed. In the case of Cuckoo search the rule (13) is used to perform the transition, where $\hat{x}^i$ is the complement of x^i. For the Black Hole the rule (12) is used, where x^i_{bh} is the position of the best solution obtained after the last perturbation. Finally, each solution is repaired using the repair operator shown in Algorithm 1.

$$x^i(t+1) := \begin{cases} x^i_{bh}(t), & \text{if } rand < P_{tg}(x^i) \\ x^i(t), & \text{otherwise} \end{cases} \tag{12}$$

$$x^i(t+1) := \begin{cases} \hat{x}^i(t), & \text{if } rand < P_{tg}(x^i) \\ x^i(t), & \text{otherwise} \end{cases} \tag{13}$$

4.3 Repair operator

In each movement performed by operators: transition ranking , local search and perturbation, it is possible to generate solutions that are infeasible. Therefore, each candidate solution must be checked and modified to meet every constraint. This verification and subsequent repairing is performed using the measure defined in Section 4.1 (8). The procedure is shown in Algorithm 1. As input the repair operator receives the solution S_{in} to repair, and the output of the repair operator gives the repaired solution S_{out}. As a first step, the repair algorithm asks whether the solution needs to be repaired. In the case that the solution needs repair, a weight is calculated for each element of the solution using the measure defined in (8). The element of the solution with the largest measure is returned and removed from the solution. This element is named s_{max}. This process is iterated until our solution does not require repair. The next step is to improve the solution. The (8) is again used for obtaining the element with the smallest measure that meets the constraints s_{min} and add s_{min} to the solution. In the case of absence of elements, empty is returned. The algorithm iterates until there are no elements that satisfy the constraints.

Algorithm 1 Repair Algorithm

Function Repair(S_{in})
Input Input solution S_{in}
Output The Repair solution S_{out}
$S \leftarrow S_{in}$
while needRepair(S) == True **do**
    $s_{max} \leftarrow$ getMaxWeight(S)
    $S \leftarrow$ removeElement(S, s_{max})
end while
state $\leftarrow$ False
while state == False **do**
    $s_{min} \leftarrow$ getMinWeight(S)
    if s_{min} == $\emptyset$ **then**
        state $\leftarrow$ True
    else
        $S \leftarrow$ addElement(S, s_{min})
    end if
end while
$S_{out} \leftarrow S$
return S_{out}

4.4 Local search operator

When the algorithm KMTR finds a solution having a fitness value higher than the best solution obtained until now, KMTR makes a call to the local search operator. This algorithm aims to perform a local search to improve the quality of the solution. The main idea of the local search operator is to add an element of the ones that are not in the solution, then to perform the repair of the solution using the repair operator. Finally it is evaluated if a better solution is obtained. To this new solution (S), another element of the complement is added, repeating the repair and comparison operations. This is iterated until incorporated all elements that were not in the initial solution. Subsequently we consider again our initial solution (S_{in}), An element is removed from it, then the solution(S) is repaired and compared. In this case, it is iterated over all elements of the solution. The pseudo-code is shown in Algorithm 2.

Algorithm 2 Local search Algorithm

Function LocalSearch(S_{in})
Input Input solution S_{in}
Output The improved local solution S_{out}
$S \leftarrow S_{in}$
$S \leftarrow S_{opt}$
for i=1 in complement of S_{in} **do**
  $S \leftarrow$ addElement(S,i)
  $S \leftarrow$ RepairOperator(S)
  if *Fitness(S) > Fitness(S_{opt})* **then**
    $S_{opt} \leftarrow S$
  end if
end for
$S \leftarrow S_{in}$
for i=1 in S **do**
  $S \leftarrow$ removeElement(S,i)
  $S \leftarrow$ RepairOperator(S)
  if *Fitness(S) > Fitness(S_{opt})* **then**
    $S_{opt} \leftarrow S$
  end if
end for
$S_{out} \leftarrow S_{opt}$
return S_{out}

4.5 Perturbation operator

The k-means transition ranking operator is responsible for performing the movements to find the optimum. However our algorithm can get trapped in a local optimum. To exit out of this local deep optimum, the transition ranking operator is complemented by a perturbation operator. This perturbation operator makes η_ν random deletions. Later the perturbed solution is completed using the repair operator. The number η_ν is obtained by considering the total length of the solution and multiplying by the factor ν. This factor ν is a parameter of the algorithm and must be estimated . This parameter controls the strength of the perturbation. This perturbation is applied to the x_{best} and to the list of feasible solutions. The procedure is outlined in Algorithm 3

Algorithm 3 Perturbation Algorithm

Function Perturbation(S_{in}, η_ν)
Input Input solution S_{in}, strength of perturbation η_ν
Output The perturbed solution S_{out}
$S \leftarrow S_{in}$
for i=1 to η_ν **do**
  Randomly remove a element of S
end for
$S_{out} \leftarrow$ RepairOperator(S)
return S_{out}

5 Results

For an adequate evaluation of our framework, we present computational results of 270 instances of the OR-library [8]. As a starting point, the methodology for obtaining the parameters of metaheuritics and binarization is detailed. Later the analysis of the key ingredients of our framework is developed. Finally, comparisons were made with recently published algorithms that use transfer functions and Quantum approach as binarization techniques. To perform the statistical analysis in this study, the non-parametric test of wilcoxon signed-rank test and violin charts are used. The violin chart is a combination of Box Plot and Kernel Density Plot widely used in machine learning and data mining [4, 28, 32]. The analysis is performed by comparing the dispersion, median and the interquartile range of the distributions.

Benchmark instances The problems were generated by varying the number of constraints $m \in \{5, 10, 30\}$ and the number of elements $n \in \{100, 250, 500\}$. For each condition (n, m) 30 problems were generated. Each set of 30 problems is divided into groups associated with the capabilities b_i where $b_i = t \times \sum_{j \in N} a_{ij}$. $t \in \{0.25, 0.5, 0.75\}$ corresponds to the tightness ratio. Each problem group used the following cb.n.m nomenclature. Where n corresponds to the total number of elements, and m the number of constraints.

For the execution of the instances we use a PC with windows 10, Intel Core i7-4770 processor with 16GB in RAM, and programmed in Python 2.7. The techniques used in the binarization were Black Hole and Cuckoo search which were named KMTR-BH and KMTR-Cuckoo respectively.

5.1 Parameter setting

As a starting point, we describe the methodology used to perform the estimation of parameters for each of the metaheuristics used. The parameters settings are shown in Tables 1 and 2. The Value column indicates the final value used by the parameter. The Range column indicates

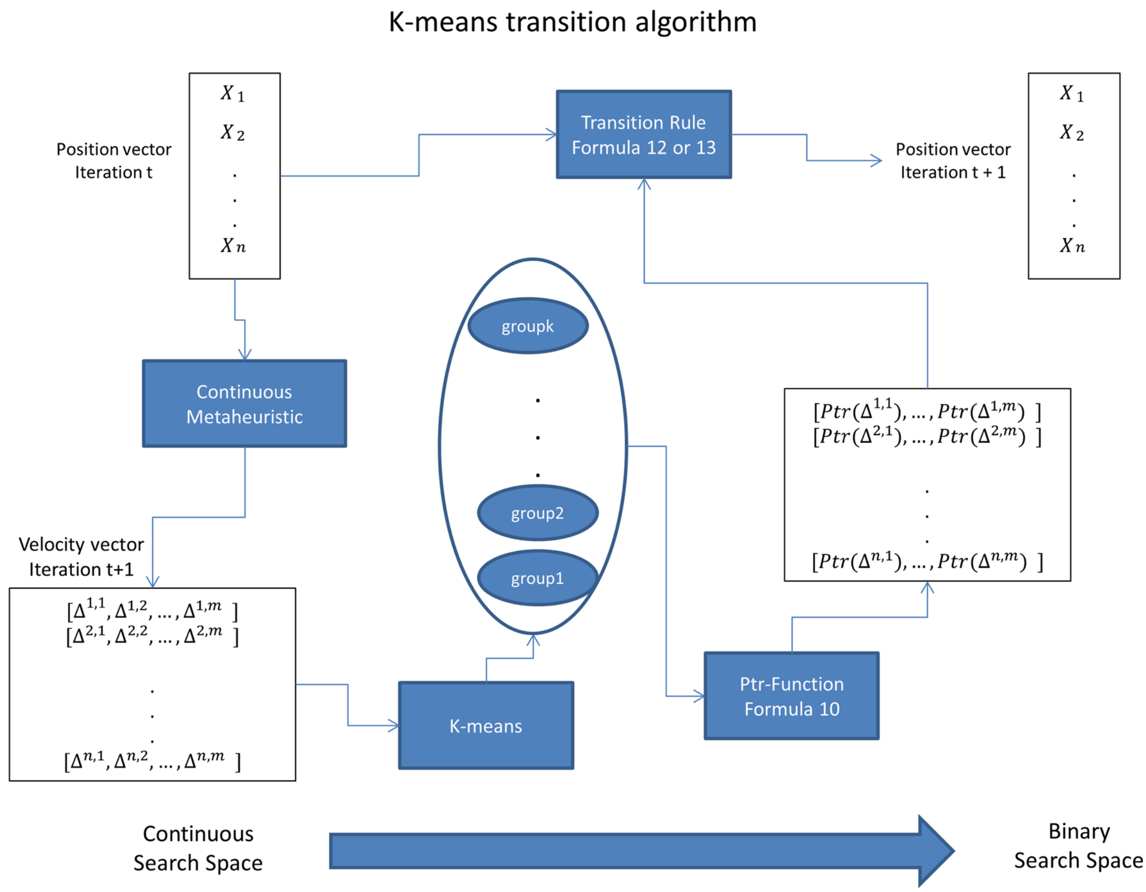


Fig. 4 Mapping the continuous search space to a discrete search space

the scanned values to obtain the final setting. To perform the scan settings, three problems were chosen for each of the groups cb.100.5, cb.250.5, cb.500.5, cb.100.10, cb.250.10,cb.500.10, cb.100.30, cb.250.30,cb.500.30. In each problem and configuration, the KMTR algorithm was executed 10 times for each metaheuristics and combination. Four measures were defined for the setting selection of the algorithm.

Table 1 Setting of parameters for black hole algorithm

Parameters	Description	Value	Range
α	Transition probability coefficient	0.1	[0.08, 0.1, 0.12]
β	Transition separation coefficient	1	
ν	Coefficient for the perturbation operator	3%	[3, 4, 5]
N	Number of particles	20	[15, 20, 25]
G	Number of transition groups	8	[7,8,9,10]
Iteration number	Maximum iterations	800	[800]

1. The percentage deviation of the best value obtained in the ten executions compared with the best known value, see (14)

$$bSolution = 1 - \frac{KnownBestValue - BestValue}{KnownBestValue} \tag{14}$$

Table 2 Setting of parameters for cuckoo search algorithm

Parameters	Description	Value	Range
α	Transition probability coefficient	0.1	[0.08, 0.1, 0.12]
β	Transition separation coefficient	1	
ν	Coefficient for the perturbation operator	3%	[2, 3, 4]
N	Number of nest	20	[15, 20, 25]
G	Number of transition groups	8	[7,8,9,10]
γ	Step length	0.01	[0.009,0.01,0.011]
κ	Levy distribution parameter	1.5	[1.4,1.5,1.6]
Iteration number	Maximum iterations	800	[800]

2. The percentage deviation of the worst value obtained in the ten executions compared with the best known value, see (15)

$$wSolution = 1 - \frac{KnownBestValue - WorstValue}{KnownBestValue} \quad (15)$$

3. The percentage deviation of the average value obtained in the ten executions compared with the best known value, see (16)

$$aSolution = 1 - \frac{KnownBestValue - AverageValue}{KnownBestValue} \quad (16)$$

4. The convergence time for the best value in each experiment normalized according to the (17)

$$nTime = 1 - \frac{convergenceTime - minTime}{maxTime - minTime} \quad (17)$$

Because we have four distinct measures, we used the area of the radar charts to evaluate the best performance configuration. Radar charts are widely used in data mining and bioinformatic [2, 67]. Each axis of the chart corresponds to one of the measures defined above. These measures take values between 0 and 1 where 1 is the best value that can be obtained. Therefore the comparison between the different configurations is the area that contains the results of the four measures. The larger the area, the better the associated configuration performs. In Fig. 5 the four best configuration results are shown as an example for the KMTR-BH algorithm.

5.2 Insight of KMTR framework

In this section we investigate some important ingredients of KMTR to get insight into the behavior of the proposed algorithm. To carry out this comparison the first 10 problems of the set cb.5.250 of the OR library and KMTR-BH were

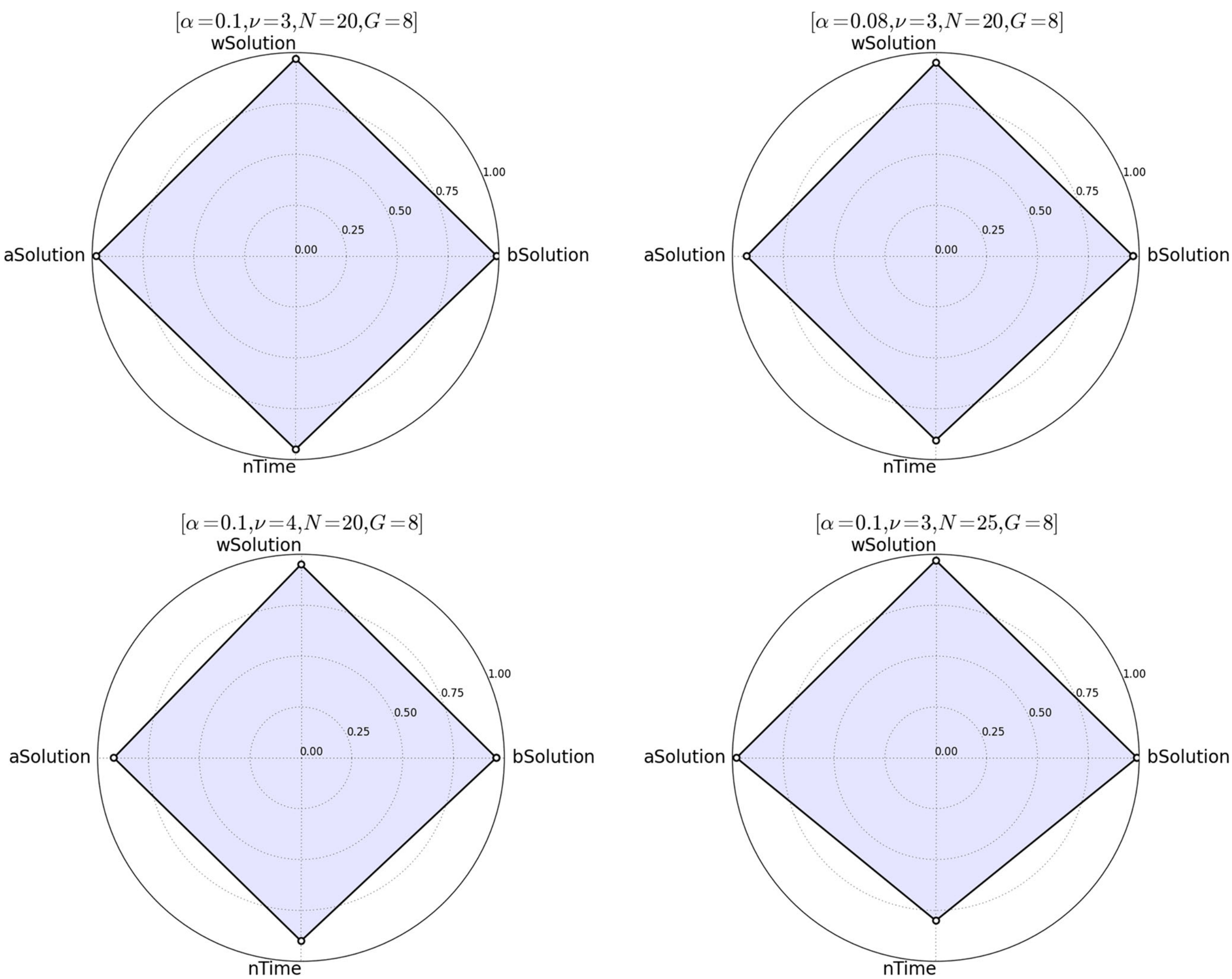


Fig. 5 Radar graphics examples for the black hole configuration

Table 3 Evaluation of element weighting

Set	Best Known	Best *KMTR-BH-AO*	Best KMTR-BH	Avg *KMTR-BH-AO*	Avg KMTR-BH
cb.5.250-0	59312	59225	59225	59141.6	59150.2
cb.5.250-1	61472	61428	61472	61342.3	61356.2
cb.5.250-2	62130	62032	62074	61946.7	61961.0
cb.5.250-3	59463	59446	59446	59304.7	59318.6
cb.5.250-4	58951	58914	58951	58799.3	58825.9
cb.5.250-5	60077	60015	60056	59919.3	59945.3
cb.5.250-6	60414	60355	60355	60286.4	60289.1
cb.5.250-7	61472	61383	61383	61319.4	61341.8
cb.5.250-8	61885	61885	61885	61747.8	61758.4
cb.5.250-9	58959	58866	58866	58785.1	58786.9
Average	60413.5	60354.9	60371.3	60259.3	60273.3
p-value				2.11 e-05	

chosen. The contribution of operators perturbation, k-means transition ranking and local search on the final performance of the algorithm was studied. To compare the distributions of the results of the different experiments we use a violin chart. The horizontal axis corresponds to the problems. The Y axis uses the measure % - Gap defined in (18)

$$\% - Gap = 100\frac{BestKnown - SolutionValue}{BestKnown} \quad (18)$$

Furthermore, a non-parametric Wilcoxon signed-rank test is carried out to determine if the results of KMTR with respect to other algorithms have significant difference or not.

5.2.1 Evaluation of the element weighting

To evaluate the contribution of the element weighting to the performance of the algorithm we compared the KMTR-BH algorithm which includes Dynamic Average Occupancy given in (8) with KMRT-BH-AO algorithm which uses the average occupancy given in (7). The results are shown in Table 3 and in Fig. 6. The table shows that for both the best value and the average KMTR-BH is higher than KMTR-

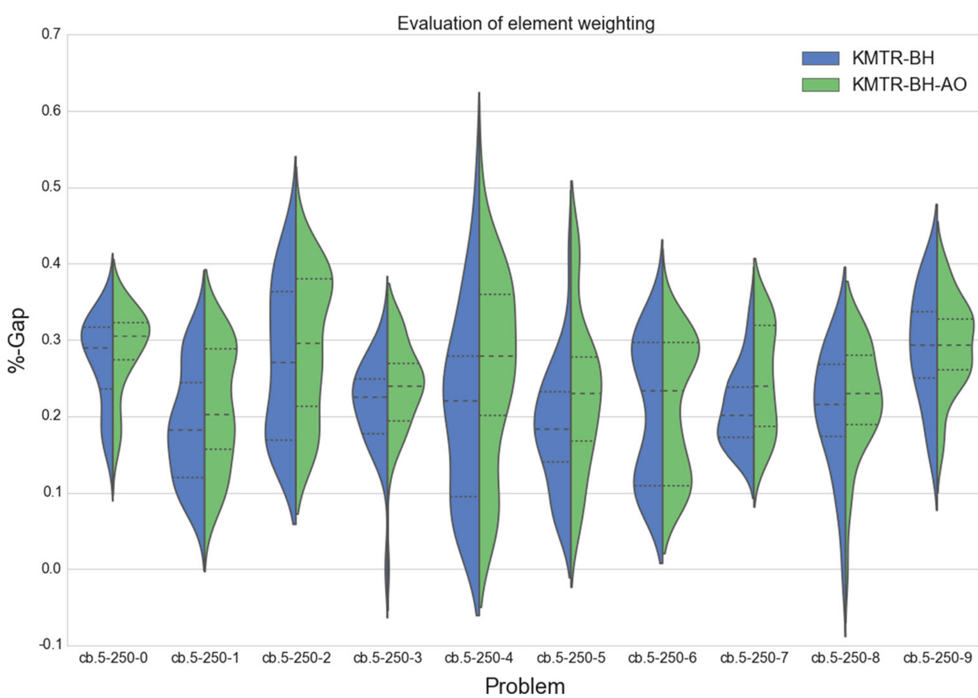


Fig. 6 Evaluation of element weighting

Table 4 Evaluation of perturbation operator

Set	Best known	Best *KMTR-BH-wp*	Best KMTR-BH	Avg *KMTR-BH-wp*	Avg KMTR-BH
cb.5.250-0	59312	59211	59225	59112.7	59150.2
cb.5.250-1	61472	61409	61472	61314.7	61356.2
cb.5.250-2	62130	62032	62074	61901.7	61961.0
cb.5.250-3	59463	59330	59446	59218.6	59318.6
cb.5.250-4	58951	58881	58951	58686.4	58825.9
cb.5.250-5	60077	60015	60056	59905.6	59945.3
cb.5.250-6	60414	60348	60355	60218.1	60289.1
cb.5.250-7	61472	61383	61383	61283.0	61341.8
cb.5.250-8	61885	61829	61885	61712.8	61758.4
cb.5.250-9	58959	58826	58866	58727.1	58786.9
Average	60413.5	60326.4	60371.3	60208.1	60273.3
p-value				3.73 e-04	

BH-AO. In Fig. 6, distributions of results are compared. It is observed that the dispersions are quite similar, however in the interquartile ranges KMTR-BH is superior in most cases. The Wilcoxon test indicates that this difference is significant therefore in the following experiments the Dynamic Average Occupancy will be used as element weight.

5.2.2 Evaluation of perturbation operator

This section aims to investigate the contribution of perturbation operator in the result of our KMTR-BH algorithm. To do this research, the KMTR-BH algorithm is configured without the perturbation operator. This algorithm is denoted by *KMTR-BH-wp*. The *KMTR-BH-wp* algorithm is compared with our perturbation operator version KMTR-BH. In both cases default parameters are used (Section 5.1). The results are shown in Table 4 and Fig. 7.

When we compare the results of the Table 4. We note that KMTR-BH is consistently better to obtain best values and averages than *KMTR-wp*. A Wilcoxon statistical test is performed to determine the difference between distributions of averages of both algorithms. The result indicates the distributions of the averages differ ($p - values < 0.05$). The results of the difference between the distributions are

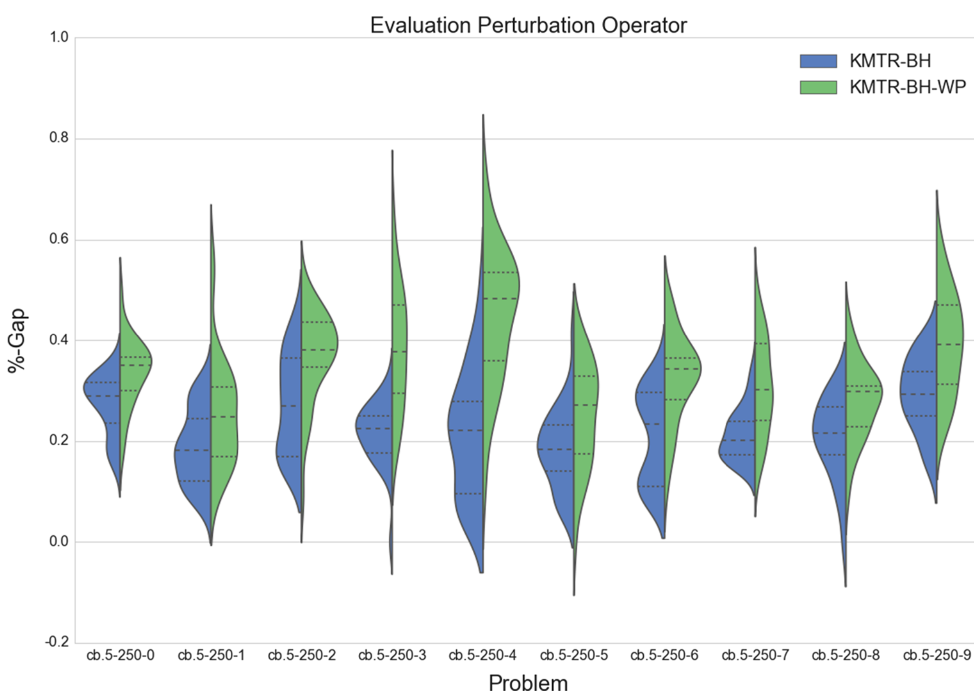


Fig. 7 Evaluation of perturbation operator

Table 5 Evaluation of k-means transition operator

Set	Best Known	Best *05.pe*	Best KMTR	Best *05.wpe*	Best *wpe*	Avg *05.pe*	Avg KMTR	Avg *05.wpe*	Avg *wpe*
cb.5.250-0	59312	59211	59225	59158	59158	59132.1	59150.2	59071.8	59123.5
cb.5.250-1	61472	61435	61472	61409	61381	61324.6	61356.2	61288.3	61350.7
cb.5.250-2	62130	62036	62074	61969	61969	61894.4	61961.0	61801.6	61925.5
cb.5.250-3	59463	59367	59446	59365	59365	59257.8	59318.6	59136.1	59260.5
cb.5.250-4	58951	58914	58951	58883	58830	58725.6	58825.9	58693.6	58777.1
cb.5.250-5	60077	60015	60056	59990	59976	59904.6	59945.3	59837.8	59946.6
cb.5.250-6	60414	60355	60355	60348	60349	60208.2	60289.1	60230.6	60327.0
cb.5.250-7	61472	61436	61383	61407	61407	61290.8	61341.8	61233.9	61337.0
cb.5.250-8	61885	61829	61885	61790	61782	61737.1	61758.4	61644.9	61746.6
cb.5.250-9	58959	58832	58866	58822	58787	58769.1	58786.9	58653.7	58738.2
Average	60413.5	60343	60371.3	60314.1	60300.4	60224.4	60273.3	60159.2	60253.3
p-value						2.67 e-06		1.54 e-07	

shown in Fig. 7. In the violin chart, the median value and the interquartile range of *KMTR-BH-wp* are displaced toward larger values of the %-Gap indicator. This suggests that the perturbation operator contributes to get better values. Moreover when we observe the dispersion of the distributions, it is observed that only problems 1 and 3, *KMTR-BH-wp* have a greater dispersion than the KMTR-BH case.

5.2.3 Evaluation of k-means transition ranking operator

To evaluate the contribution of the k-mean transition ranking operator to the final result we designed a random operator. This random operator executes the transition with a fixed probability (0.5) without considering the ranking of the particle. Two scenarios were established. In the first one the perturbation and local search operators are included. In the second one these operators are excluded. KMTR-BH corresponds to our standard algorithm. *05.pe* is the random variant that includes the perturbation and local search operators. *wpe* corresponds to the version with k-means transition operator without perturbation and local search operators. Finally *05.wpe* describes the random algorithm without perturbation and local search operators.

When we compared the Best Values between KMTR-BH and *05.pe* algorithms in Table 5. KMTR-BH outperforms to *05.pe* except for problem 7. However the Best Values between both algorithms are very close. In the Averages comparison, KMTR outperforms *05.pe* in all problems.

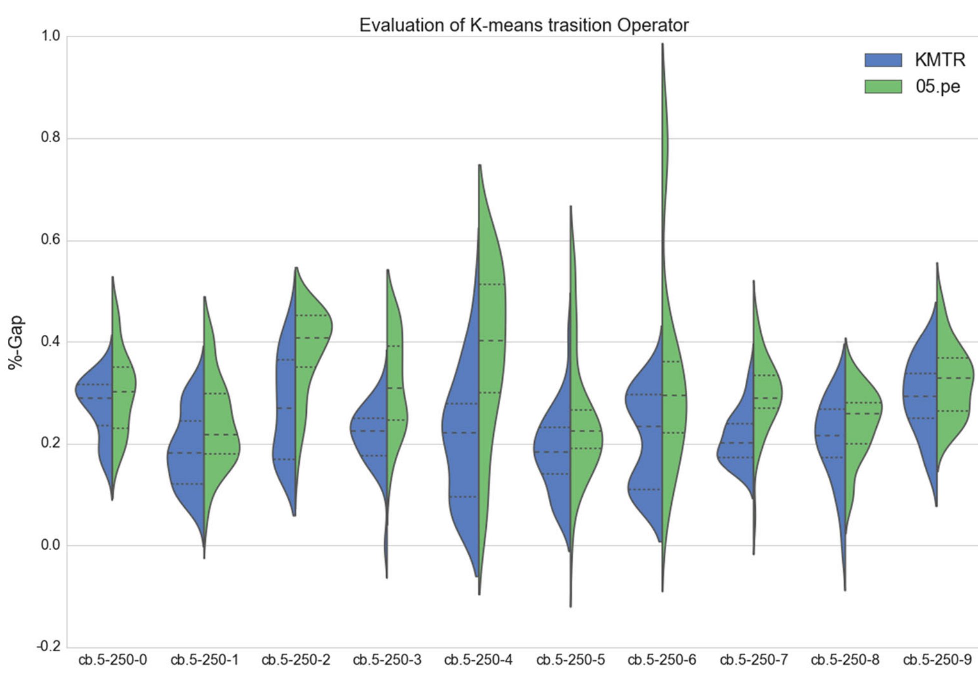
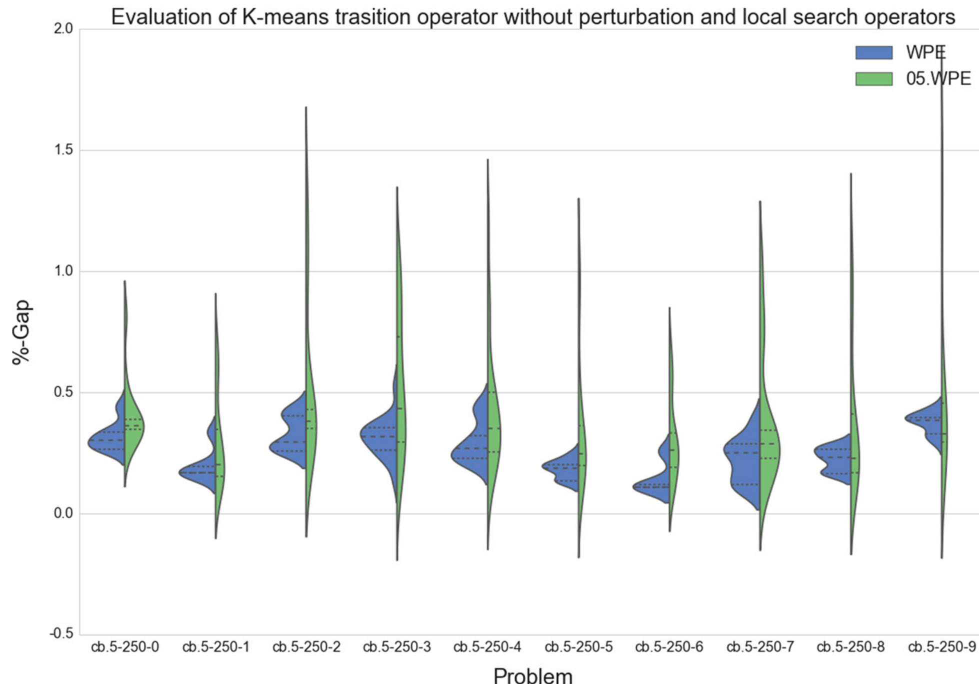


Fig. 8 Evaluation of k-means transition operator with perturbation and Local Search operators

Fig. 9 Evaluation of k-means transition operator without perturbation and local search operators



The comparison of distributions is shown in Fig. 8. We see the dispersion of the *05.pe* distributions are bigger than the dispersions of KMTR. In particular this can be appreciated in the problems 3, 4, 5, 6 and 7. Then, the k-means transition ranking operator together with perturbation operators and local search contribute to the stability of the solution. Also, the KMTR distributions are closer to zero than *05.pe* distributions, indicating that KMTR has a better performance than *05.pe*.

Our next step is trying to separate the contribution of local search and perturbation operator from the k-mean transition operator. For this, we compared the algorithms *wpe* and *05.wpe*.

When we check the Best Values shown in the Table 5, We see that the *wpe* and random *05.wpe* algorithms obtain similar results for the best indicator. However *05.wpe* slightly outperforms *wpe* in some cases. When we compare the Averages, the situation is reversed obtaining a clear supremacy of *wpe* by about *05.wpe*. Even more, when we compare the distributions shown in Fig. 9 we see that *wpe* has solutions dispersion much smaller than *05.wpe*.

5.2.4 Evaluation of local search operator

This section aims to understand the contribution of the local search operator on the final result in the optimal tracking. Again the comparison was made with the first 10 instances of the cb.5.250 group and the binarization of the Black Hole algorithm was used. We compared the KMTR-BH algorithm with the modified algorithm which did not execute

Table 6 Evaluation of local search operator

Set	Best Known	best *KMTR-BH-WLS*	best KMTR-BH	avg *KMTR-BH-WLS*	avg KMTR-BH
cb.5.250-0	59312	59211	59225	59129.6	59150.2
cb.5.250-1	61472	61377	61472	61343.8	61356.2
cb.5.250-2	62130	62002	62074	61919.8	61961.0
cb.5.250-3	59463	59317	59446	59244.6	59318.6
cb.5.250-4	58951	58914	58951	58726.2	58825.9
cb.5.250-5	60077	60007	60056	59925.2	59945.3
cb.5.250-6	60414	60348	60355	60288.4	60289.1
cb.5.250-7	61472	61382	61383	61306.8	61341.8
cb.5.250-8	61885	61829	61885	61721.4	61758.4
cb.5.250-9	58959	58822	58866	58786.3	58786.9
Average	60413.5	60320.9	60371.3	60239.2	60273.3
p-value				1.58 e-06	

the local search. To this modified algorithm, it was denoted by *KMTR-BH-WLS*. In the Table 6, the comparison is shown. KMTR-BH shows higher results than *KMTR-BH-WLS* for the Best Value and Average indicators in all tests. The statistical verification using the Wilcoxon signed rank test indicates that this difference is significant between both algorithms. When we compared the distributions through the graphic violin Fig. 10, we see that in general the distributions have similar dispersion ranges. However, the distributions of *KMTR-BH-WLS* are shifted towards greater values of %-Gap than in the case of KMTR-BH. This indicates that the contribution of our perturbation operator is to improve the final values, without affecting the dispersion of the solutions.

5.3 KMTR framework comparisons

In this section, we describe the comparisons that were made of our framework with other recently published algorithms. Three groups of problems were chosen for the comparison. cb.5.500, cb.10.500 y cb.30.500 correspond to the larger problems of the OR-library. The first algorithm corresponds to Binary Artificial Algae Algorithm (BAAA) developed by Zhang [80]. This algorithm uses V-shape transfer function as a binarization mechanism. The set of problems cb.5.500 was used for comparison. The second algorithm is a Binary differential search algorithm (BDS) developed by Liu [43]. This algorithm also uses a V-shape transfer function as a binarization mechanism. The set of problems cb.10.500 was used for comparison. Finally, in the third comparison a Hybrid Quantum Particle Swarm Optimization (QPSO*) developed by Haddar [27] was used to compare the cb.30.500 group. The comparisons were made using the Best value indicator, which corresponds to the best value obtained by the algorithm in the different executions and the Average indicator which corresponds to the average of the results obtained considering all executions. For clarity of the comparisons between the algorithms, Table 7 is incorporated. This table summarizes the results of best values and averages, where the comparisons are made considering the algorithms in pairs. In the case that both algorithms have the same value, this is not considered in the accounting.

Table 7 Summary of comparisons

	KMTR-BH		KMTR-Cuckoo	
	Best Value	Average	Best Value	Average
BAAA	$6^{BAAA}/24^{BH}$	2/26	12/18	2/22
TR-DBS	6/23	1/29	7/23	1/29
TE-DBS	22/8	10/20	22/8	9/21
QPSO*	30/0	30/0	30/0	30/0

5.3.1 Comparison with BAAA

In this section we evaluate the performance of our KMTR-framework with the algorithm BAAA developed in [80].

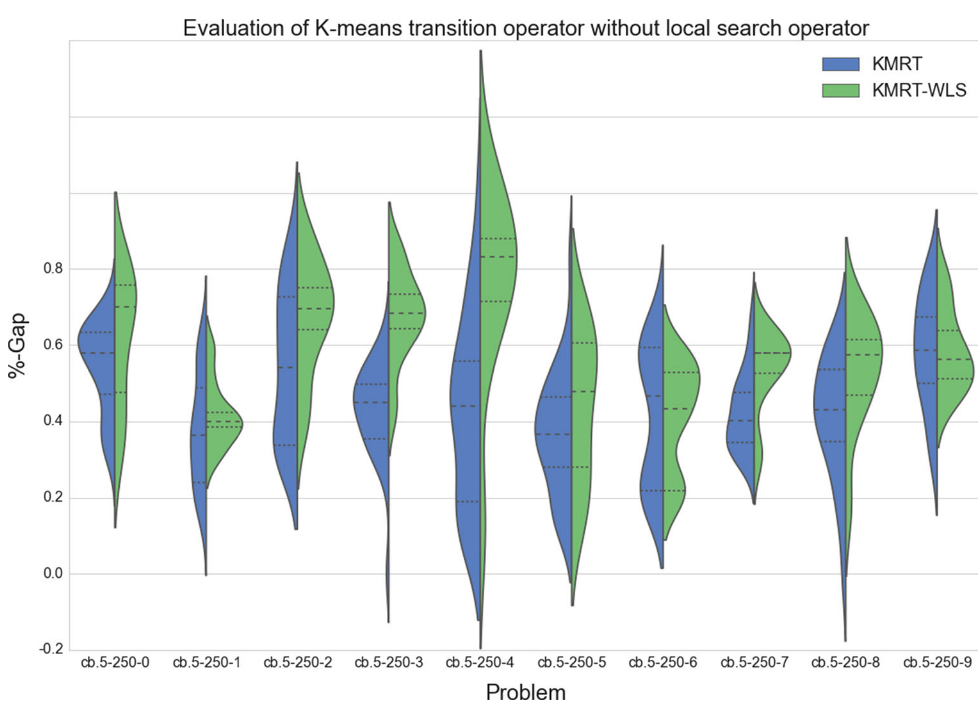


Fig. 10 Evaluation of k-means transition operator without local search operator

Table 8 OR-Library benchmarks MKP cb.5.500

Instance	Best Known	BAAA Best	Avg	std	KMTR-BH Best	Avg	Time(s)	std	KMTR-Cuckoo Best	Avg	Time(s)	std
0	120148	120066	120013.7	21.57	**120096**	120029.9	475	25.3++(2.7)	120082	**120036.8**	256	25.5++(3.7)
1	117879	117702	117560.5	11.4	**117730**	**117617.5**	512	55.8++(5.5)	117656	117570.6	278	57.3+(0.4)
2	121131	120951	120782.9	87.96	**121039**	**120937.9**	366	50.3++(8.4)	120923	120855.1	238	39.8++(4.0)
3	120804	120572	120340.6	106.01	**120683**	**120522.8**	467	71.6++(7.8)	**120683**	120455.7	219	32.1++(5.7)
4	122319	122231	122101.8	56.95	**122280**	**122165.2**	429	50.2++(4.5)	122212	122136.4	209	39.4++(2.7)
5	122024	121957	121741.8	84.33	**121982**	**121868.7**	428	52.2++(7.0)	121946	121824.6	198	35.4++(4.9)
6	119127	119070	118913.4	63.01	**119068**	**118950.0**	486	52.9++(2.4)	118956	118895.5	217	40.1−(1.3)
7	120568	120472	120331.2	69.09	120463	**120336.6**	389	45.7+(0.4)	120392	120320.4	267	43.0−(0.7)
8	121586	121052	120683.6	834.88	**121377**	**121161.9**	410	91.3++(3.1)	121201	121126.3	235	54.5++(2.9)
9	120717	120499	120296.3	110.06	**120524**	**120362.9**	397	89.0++(2.6)	120467	120335.5	213	62.2+(1.7)
Average	120630.3	120457.2	120276.5	154.5	**120524.2**	**120395.3**	435.9	58.4	120451	120355.7	233	43.0
p-value										2.43 e-4		
10	218428	218185	217984.7	123.94	**218296**	218163.7	412	50.7++(7.3)	218291	**218208.9**	340	47.8++(9.2)
11	221202	220852	220527.5	169.16	220951	220813.9	379	65.3++(8.7)	**220969**	**220862.3**	319	63.9++(10.1)
12	217542	217258	217056.7	104.95	217349	217254.3	397	51.4++(9.3)	**217356**	**217293.0**	298	53.3++(10.9)
13	223560	223510	223450.9	26.02	**223518**	223455.2	405	32.8+(06)	223516	**223455.6**	341	45.4+(0.4)
14	218966	218811	218634.3	97.52	218848	218771.5	402	44.0++(7.0)	**218884**	**218794.0**	289	49.0++(8.0)
15	220530	220429	220375.9	31.86	220441	220342.2	379	56.6−−(2.8)	**220433**	**220352.7**	269	40.8−−(2.4)
16	219989	219785	219619.3	93.01	219858	219717.9	398	60.1++(4.9)	**219943**	**219732.8**	297	47.2++(5.9)
17	218215	218032	217813.2	115.37	218010	217890.1	386	57.3++(3.3)	**218094**	**217928.7**	295	55.4++(4.9)
18	216976	**216940**	**216862.0**	32.51	216866	216798.8	468	41.2−−(6.6)	216873	216829.8	345	39.5−−(3.4)
19	219719	219602	219435.1	54.45	219631	219520.0	429	52.4++(6.2)	**219693**	**219558.9**	321	54.9++(8.8)
Average	219512.7	219340.4	219175.9	84.8	219376.8	219272.7	405.5	51.2	**219405.2**	**219301.1**	311.4	49.7
p-value										1.45 e-5		
20	295828	295652	295505.0	76.30	**295717**	**295628.4**	348	48.9++(7.5)	295688	295608.8	275	33.1++(6.8)
21	308086	307783	307577.5	135.94	307924	307860.6	564	55.7++(10.6)	**308065**	**307914.8**	309	59.1++(12.5)
22	299796	299727	299664.1	28.81	**299796**	**299717.8**	394	89.9++(3.1)	299684	299660.9	257	12.1−(0.6)
23	306480	306469	306385.0	31.64	**306480**	**306445.2**	437	96.1++(3.3)	306415	306397.3	285	17.1+(1.8)
24	300342	300240	300136.7	51.84	**300245**	**300202.5**	428	26.4++(6.2)	300207	300184.4	274	16.9++(4.8)
25	302571	**302492**	302376.0	53.94	302481	**302442.3**	439	24.1++(6.1)	302474	302435.6	298	24.0++(5.5)
26	301339	301272	301158.0	44.3	301284	301238.3	386	37.9++(7.5)	**301284**	**301239.7**	278	24.1++(8.9)
27	306454	306290	306138.4	84.56	306325	306264.2	468	45.4++(7.2)	**306331**	**306276.4**	286	23.8++(8.6)
28	302828	302769	302690.1	34.11	302749	**302721.4**	401	22.4++(4.2)	**302781**	302716.9	268	32.0++(3.1)
29	299910	299757	299702.3	31.66	299774	299722.7	397	34.1++(2.4)	**299828**	**299766.0**	297	45.5++(6.3)
Average	302363.4	302245.1	302133.3	57.3	302277.5	**302224.3**	426.2	48.1	**302275.7**	302220.1	282.7	28.8
p-value										3.29 e-4		

Bold represents the algorithm that had the best performance

BAAA 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 transference. The parameter τ of the tanh function was set to a value 1.5. Additionally a 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 KMTR-framework, the configurations are the same used in the previous experiments. These are described in the Tables 1 and 2. In addition, in order to

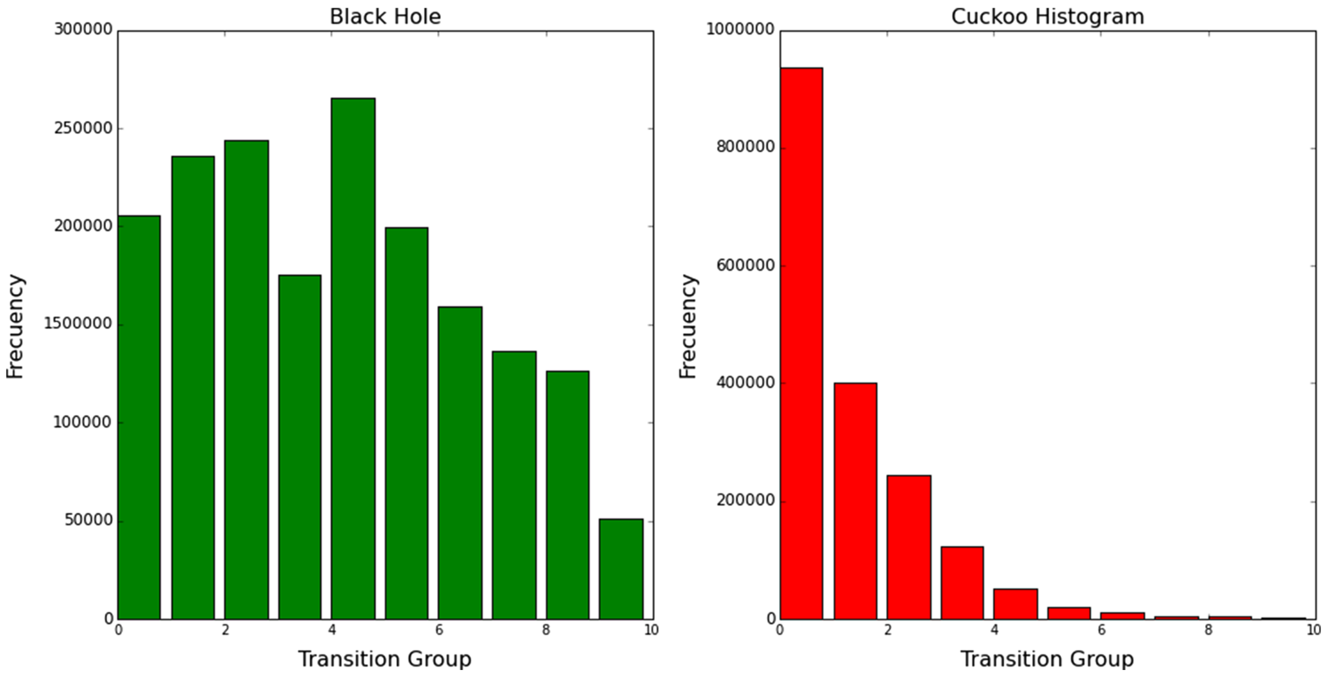


Fig. 11 Transition group histograms

determine if KMTR average is significantly different than averages obtained by BAAA, we have performed Student's t-test. The t statistic has the following form:

$$t = \frac{\hat{X}_1 - \hat{X}_2}{\sqrt{\frac{(n_1-1)SD_1^2+(n_2-1)SD_2^2}{n_1+n_2-2}\frac{n_1+n_2}{n_1 n_2}}} \tag{19}$$

Where:

$\hat{X}_1$: Average of BAAA for each instance
SD_1: Standard deviation of BAAA for each instance
n_1: number of test for BAAA for each instance
$\hat{X}_2$: Average of KMTR-BH or KMTR-Cuckoo for each instance

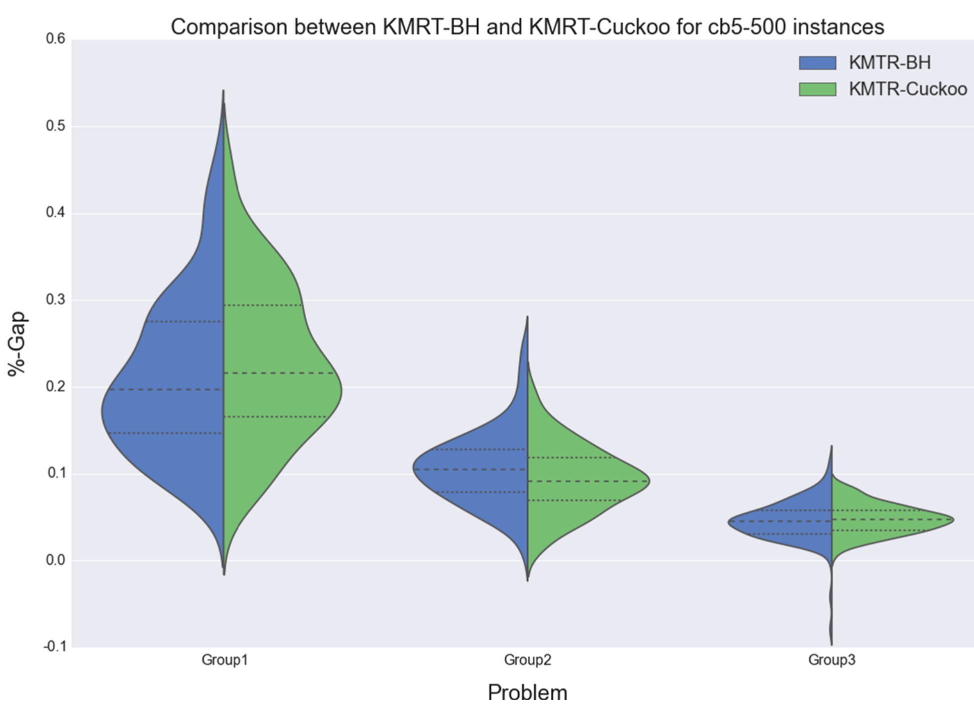


Fig. 12 Comparison between KMTR-BH and KMTR-Cuckoo for cb.5.500 instances

Table 9 OR-Library benchmarks MKP cb.10.500

Instance	Best Known	TR-DBS Best	Avg	TE-DBS Best	Avg	KMTR-BH Best	Avg	Time(s)	KMTR-Cuckoo Best	Avg	Time
0	117821	114716	114425.4	**117811**	**117801.2**	117558	117293.0	584	117509	117302.8	467
1	119249	119232	**119223.0**	**119249**	118024.0	119232	118980.8	547	119072	118936.2	437
2	119215	119215	117625.6	**119215**	117801.4	118940	**118840.5**	548	119039	118723.0	423
3	118829	**118813**	117625.8	**118813**	117801.2	118598	**118516.2**	539	118586	118433.9	428
4	116530	114687	114312.4	**116509**	114357.2	116186	**116095.5**	532	116312	116013.8	410
5	119504	119504	112503.7	**119504**	117612.8	119257	**119113.2**	437	119257	119065.7	439
6	119827	116094	115629.1	**119827**	**119827.4**	119691	119556.7	463	119663	119506.6	419
7	118344	116642	115531.9	**118301**	117653.3	118016	**117907.3**	437	118058	117760.3	436
8	117815	114654	114204.0	**117815**	115236.4	117550	**117363.0**	483	117550	117235.0	417
9	119251	114016	113622.8	**119231**	118295.1	118896	**118739.0**	485	118962	118514.2	414
Average	118638.5	116757.3	115470.3	**118627.5**	117441	118392	**118240.5**	505.5	118400.8	118149.1	429
p-value									1.93 e-5		
10	217377	209191	208710.2	**217377**	212570.3	216990	216892.0	446	217126	**216892.0**	394
11	219077	**219077**	217277.2	**219077**	218570.2	218672	218592.0	437	218872	**218592.4**	389
12	217847	210282	210172.3	217377	212570.4	217447	217358.8	428	**217573**	**217542.3**	412
13	216868	209242	206178.6	**216868**	**216868.9**	216570	216484.5	457	216570	216469.9	394
14	213873	207017	206656.0	207017	206455.0	**213474**	**213374.0**	427	213474	213363.5	378
15	215086	204643	203989.5	**215086**	**215086.0**	214761	214638.7	420	214829	214702.6	356
16	217940	205439	204828.9	**217940**	**217940.5**	217583	217484.2	438	217629	217567.1	395
17	219990	208712	207881.6	**219984**	209990.2	219589	219496.8	428	219675	**219554.4**	374
18	214382	210503	209787.6	210735	211038.2	214015	213862.7	429	**214045**	**213939.4**	389
19	220899	205020	204435.7	**220899**	219986.8	220488	220391.3	436	220582	**220515.1**	369
Average	217333.9	208912.6	207991.7	216236	214107.6	216958.9	216857.5	434.6	**217037**	**216913.8**	385
p-value									4.85 e-4		
20	304387	**304387**	302658.8	304387	**304264.5**	304102	303991.6	419	304116	304019.1	344
21	302379	**302379**	301658.6	302379	**302164.4**	302138	302078.2	441	302263	302156.3	328
22	302417	290931	290859.9	**302416**	302014.6	302103	301968.2	438	302118	**302061.6**	349
23	300784	290859	290021.4	291295	291170.6	300542	300480.4	429	**300566**	**300498.3**	358
24	304374	289365	288950.1	**304374**	**304374.0**	304267	304168.7	427	304229	304187.7	338
25	301836	292411	292061.8	**301836**	**301836.0**	301730	301461.8	420	301445	301332.1	324
26	304952	291446	290516.2	291446	291446.0	304833	304778.1	413	**304905**	**304814.6**	348
27	296478	293662	293125.5	295342	294125.5	296263	296194.0	441	**296361**	**296288.9**	364
28	301359	285907	285293.4	288907	287923.4	301085	301026.0	426	**301085**	**301031.2**	326
29	307089	290300	289552.4	295358	290525.2	306881	306786.6	419	**306881**	**306786.6**	351
Average	302605.5	293164.7	292469.8	297774	296984.4	302394.4	302293.3	427.3	**302396.9**	**302317.6**	343
p-value									5.27 e-4		

Bold represents the algorithm that had the best performance

SD_2: Standard deviation KMTR-BH or KMTR-Cuckoo for each instance

n_2: number of test for KMTR-BH or KMTR-Cuckoo for each instance

The t values can be positive, neutral, or negative. The double positive value (++) of t indicates that KMTR is significantly better than BAAA. In the opposite case (−−), KMTR obtains significant worse solutions. If t is single positive (+), KMTR shows to be better but not significantly. On the other hand, if the result is single negative (−), KMTR 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. For the case of comparative summary shown in Table 7, we consider only the results that have significance.

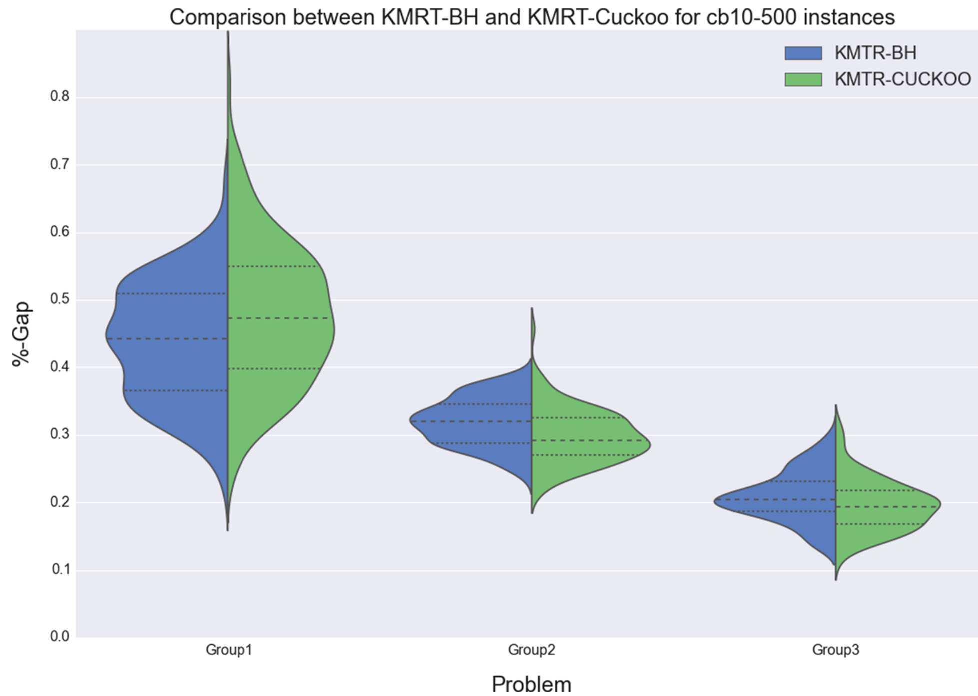


Fig. 13 Comparison between KMTR-BH and KMTR-Cuckoo for cb.10.500 instances

The results are shown in Table 8. The comparison was performed for the set cb.5.500 of the OR-library. The results for KMTR-BH and KMTR-Cuckoo were obtained from 30 executions for each problem. In black, the best results are marked for both indicators the Best Value and the Average. We must emphasize that although BAAA has quite good results, KMTR-BH and KMTR-Cuckoo exceed it in practically all problems. In the Best Value indicator, BAAA was higher in three instances, KMTR-BH in 15 and KMTR-Cuckoo in 13. It should be noted that in instance 4 KMTR-BH and KMTR-Cuckoo obtained the same value. In the averages indicator BAAA was higher in 1 instance, KMTR-BH in 15 and KMTR-Cuckoo in 14.

By observing the execution times, we see that Cuckoo has better runtime than BH. Inquiring about the causes of this difference, a sampling of the displacements (Δ) of both metaheuristics was considered. With these sampling, histograms were constructed to quantify the amount of displacements assigned to the different transition groups obtained by k-means. The result is shown in Fig. 11. In the case of Black Hole, the distribution of transition groups is far more homogeneous than in Cuckoo. In Cuckoo most of the transitions are concentrated in the first 3 groups. This property has as a consequence that Cuckoo has fewer transitions than BH.

Finally we compare KMTR-BH y KMTR-Cuckoo. For comparison, we organized the problems into three groups 0 to 9, 10 to 19 and 20 to 29. The Wilcoxon test shown in Table 8 and violin charts shown in Fig. 12 were used for comparison. In all three groups there are differences between distributions, KMTR-BH obtained better solutions for the first group and KMTR-Cuckoo in the second. Although the Wilcoxon test and the transition histograms shown in Fig. 11 indicates that the distributions of the KMTR-BH and KMTR-Cuckoo results are different, when we observe the distributions in detail with the violin graph, we observe that they are similar in form, values and dispersion. The main differences that distinguish both distributions correspond to the median values and interquartile range.

5.3.2 Comparison with DBS

In this section we evaluate the performance of our KMTR-framework with the algorithms TR-DBS (Tanh Random) and TE-DBS (Tanh Elitist) developed in [43]. DBS uses transfer functions as a general mechanism of binarization. In particular DBS used the tanh $= \frac{e^{\tau|x|}-1}{e^{\tau|x|}+1}$ function to perform the transference. The parameter τ of the tanh function was set to a value 2.5. As maximum number of iterations DBS used 10000. 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 KMTR-framework, the configurations are the same used in the previous experiments. These are described in the Tables 1 and 2.

The results are shown in Table 9. The comparison was performed for the set cb.10.500 of the OR library. The results for KMTR-BH and KMTR-Cuckoo were obtained from 30 executions for each problem. The best results for the Best Value and Average indicators are marked in black.

Table 10 OR-Library benchmarks MKP cb.30.500

Instance	Best Known	QPSO* Best	QPSO* Avg	KMTR-BH Best	KMTR-BH Avg	KMTR-BH Time(s)	KMTR-BH std	KMTR-Cuckoo Best	KMTR-Cuckoo Avg	KMTR-Cuckoo Time(s)	KMTR-Cuckoo std
0	116056	**115991**	**115906.0**	115449	115236.6	638	151.5	115526	115341.7	464	81.8
1	114810	**114684**	**114661.0**	114352	114172.2	610	124.0	114405	114291.5	429	77.6
2	116741	**116712**	**116642.5**	116158	116065.5	578	78.8	116256	116093.8	489	78.5
3	115354	**115354**	**115062.5**	114739	114574.4	570	125.0	114782	114673.9	578	52.1
4	116525	**116435**	**116378.5**	115994	115881.2	594	148.8	115995	115872.7	548	82.9
5	115741	**115594**	**115583.5**	115244	115149.5	610	111.5	115342	115143.0	486	114.6
6	114181	**113987**	**113936.5**	113593	113433.7	629	158.5	113712	113527.5	528	112.6
7	114348	**114184**	**114135.5**	113610	113522.0	649	89.7	113626	113516.5	519	79.5
8	115419	**115419**	**115271.0**	114705	114684.2	628	62.4	114822	114633.4	589	85.4
9	117116	**116909**	**116909.0**	116382	116374.5	638	22.5	116467	116338.6	519	90.0
Average	115629,1	115526,9	115448,6	115022,6	114909,3	614.4	107,2	115093,3	114943,26	514.9	85.5
p-value									1.86 e-7		
10	218104	**218068**	**218068.0**	217607	217574.8	573	34.4	217776	217619.2	539	72.8
11	214648	**214626**	**214546.5**	214110	214089.5	539	61.5	214110	214002.7	530	58.9
12	215978	**215839**	**215839.0**	215580	215506.5	528	59.3	215638	215494.8	486	62.7
13	217910	**217816**	**217816.0**	217201	217136.0	568	55.0	217301	217215.8	549	54.0
14	215689	**215544**	**215544.0**	215036	214974.7	563	32.7	215116	214992.6	520	70.8
15	215919	**215753**	**215753.0**	215326	215223.6	510	130.0	215408	215219.7	269	112.1
16	215907	**215789**	**215784.5**	215516	215449.1	529	46.7	215576	215486.9	538	55.7
17	216542	**216387**	**216387.0**	215999	215981.6	521	35.5	216057	216002.6	549	27.3
18	217340	**217217**	**217211.0**	216882	216867.8	534	27.3	217013	216886.2	520	72.4
19	214739	**214739**	**214686.5**	214194	214127.3	542	46.2	214332	214127.3	517	46.2
Average	216277.6	216177.8	216163.5	215745.1	215693.1	540.7	52.8	215818.9	215704.8	501.7	63.3
p-value									5.13 e-5		
20	301675	**301643**	**301635.0**	301343	301200.8	562	59.2	301343	301241.0	592	59.6
21	300055	**299965**	**299963.5**	299636	299556.8	536	53.9	299720	299579.9	538	70.6
22	305087	**305038**	**305038.0**	304850	304774.5	569	38.0	304852	304748.2	549	62.0
23	302032	**301982**	**301982.0**	301658	301536.7	567	61.7	301645	301583.2	584	50.5
24	304462	**304346**	**304346.0**	304186	304082.4	578	61.5	304186	304106.1	529	53.5
25	297012	**296892**	**296892.0**	296450	296413.0	546	18.0	296521	296420.5	520	35.6
26	303364	**303287**	**303287.0**	302917	302841.6	548	68.9	302941	302843.8	519	71.6
27	307007	**306915**	**306915.0**	306616	306450.7	542	68.8	306616	306451.3	502	69.1
28	303199	**303169**	**303169.0**	302636	302550.3	563	49.7	302791	302565.8	510	85.4
29	300572	**300449**	**300449.0**	300170	300061.5	549	63.9	300170	300063.8	531	65.8
Average	302446.5	302368.6	302367.6	302046,2	301946.8	556	54.4	302078.5	301960.3	537.4	62.4
p-value									4.19 e-5		

Bold represents the algorithm that had the best performance

When we compare the four algorithms, we see that TE-DBS obtained the biggest amount of Best Values with a total of 20 of the 30 problems. It was followed by KMTR-Cuckoo with 7, then TR-DBS with 3 and KMTR-BH with 0. When the average indicator is analyzed, the situation is different. KMTR-Cuckoo obtained 12, then TE-DBS with 9, KMTR-BH with 7 and finally TR-DBS with 4. When we compare the Average indicator by groups of problems, where group 1 corresponds to problems 0-9, group two problems 10-19 and group 3 problems 20-29, KMTR-Cuckoo scored better in Groups 2 and 3 for both Best Value and Average, TE-DBS for Best Value in Group 1 and KMTR-BH for Group 1 in Average indicator. This indicates that although the TR-DBS and TE-DBS algorithms obtain high Best Values, these algorithm are not consistent in obtaining them, considering that the

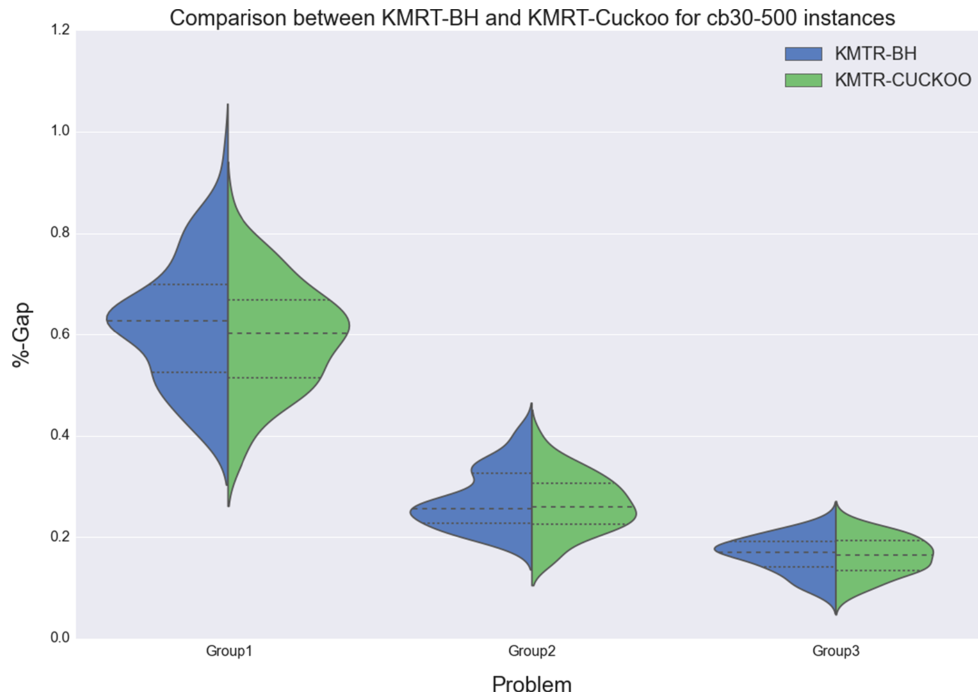


Fig. 14 Comparison between KMTR-BH and KMTR-Cuckoo for cb.30.500 instances

number of maximum iterations is 10000. The average execution times in TR-DBS and TE-DBS cases are over 5000 (s) and 6000 (s) respectively, where KMTR-BH and KMTR-Cuckoo are below 600(s). The calculation was made on equivalent computers.

Finally we compare KMTR-BH and KMTR-Cuckoo, considering the three previously defined groups. The Wilcoxon test shown in Table 9 and violin charts shown in Fig. 13 were used for comparison. In all three groups there are differences between distributions, KMTR-BH obtained better solutions for the first group and KMTR-Cuckoo in the second and third groups. Although the Wilcoxon test indicates that the distributions of the KMTR-BH and KMTR-Cuckoo results are different, when we observe the distributions in detail with the violin graph, we observe that they are similar in form, values and dispersion. The main differences that distinguish both distributions correspond to the median values and interquartile range.

*5.3.3 Comparison with QPSO**

In this section, we compare the performance of our KMTR-framework with the Quantum PSO (QPSO*) algorithm developed by [27]. This QPSO* algorithm additionally uses a local search method and a repair algorithm based on the notion of the pseudo-utility ratio. The algorithm was coded in C language, and experimental tests were performed on a Personal Computer with a 2.2 GHz Core 2 Duo processor and 3GB RAM. The number of iterations were 500, and the number of executions were 30. For our framework, the configuration was the same used in the previous experiments, described in the Tables 1 and 2.

The comparison was made using the set of problems cb.30.500 from the OR-library. The results are shown in the Table 10. In this case the superiority of QPSO* compared to our binarizations was complete in both indicators, Best Value and Average. Considering groups 0-9,10-19 and 20-29, we calculate the difference between QPSO* and our binarizations, for the Best Value and Average indicators. The maximum difference corresponds to group 1, where the comparison of the QPSO* Best Value indicator with KMTR-BH has a 0.43% deviation and KMTR-Cuckoo a 0.37%. For the case of the average indicator, group 1 obtained the difference of 0.46% for KMTR-BH 0.43% for KMTR-Cuckoo. In group 2, the differences for the Best value were 0.20 and 0.16. For the Average differences were 0.22% and 0.21% respectively. Finally for group 3 the differences in the Best Value were of 0.1% and 0.09%. For the average, 0.14% and 0.13%.

When we compared KMTR-BH and KMTR-Cuckoo, the results of the test Wilcoxon shown in the Table 10, indicate that there are differences between them. In the Fig. 14, their distributions are compared. It is observed that the difference is mainly due to the values of their medians and interquartile ranges. The dispersion and the shape of the distributions are similar in both binarizations.

Table 11 Detailed performance of KMTR-BH and KMTR-Cuckoo on OR-Library instances (based on average %-Gap)

Problem Set	KMTR-BH Average %-Gap	std	Average Time(s)	KMTR-Cuckoo Average %-Gap	std	Average Time(s)
cb.5.100.25	0.15	0.11	63.1	0.14	0.1	53.1
cb.5.100.50	0.13	0.09	61.8	0.09	0.08	51.4
cb.5.100.75	0.01	0.05	57.5	0.01	0.05	50.3
cb.5.250.25	0.23	0.13	147.6	0.25	0.14	142.8
cb.5.250.50	0.19	0.08	142.8	0.20	0.07	140.4
cb.5.250.75	0.04	0.04	142.5	0.04	0.03	137.8
cb.5.500.25	0.21	0.08	435.9	0.22	0.06	233
cb.5.500.50	0.11	0.04	405.5	0.09	0.03	311.4
cb.5.500.75	0.05	0.02	426.2	0.05	0.01	282.7
cb.10.100.25	0.41	0.31	64.1	0.36	0.18	58.2
cb.10.100.50	0.21	0.15	62.4	0.2	0.12	53.9
cb.10.100.75	0.39	0.11	59.3	0.36	0.1	54.6
cb.10.250.25	0.21	0.15	168.3	0.20	0.13	154.5
cb.10.250.50	0.11	0.07	165.8	0.1	0.06	152.5
cb.10.250.75	0.05	0.03	162.9	0.03	0.03	148.5
cb.10.500.25	0.34	0.08	505.5	0.37	0.1	429
cb.10.500.50	0.22	0.03	434.6	0.20	0.03	385
cb.10.500.75	0.11	0.03	427.3	0.09	0.03	343
cb.30.100.25	0.41	0.28	67.3	0.39	0.27	59.3
cb.30.100.50	0.28	0.17	63.6	0.26	0.16	58.7
cb.30.100.75	0.11	0.08	62.9	0.09	0.07	54.2
cb.30.250.25	0.67	0.16	164.3	0.69	0.17	152.4
cb.30.250.50	0.31	0.07	162.6	0.29	0.08	150.5
cb.30.250.75	0.16	0.03	162.5	0.14	0.02	147.4
cb.30.500.25	0.62	0.12	614.4	0.59	0.10	514.9
cb.30.500.50	0.27	0.05	540.7	0.26	0.05	501.7
cb.30.500.75	0.17	0.03	556	0.16	0.03	537.4
Average	0.24	0.1	245.6	0.22	0.09	205.7

5.3.4 Other comparisons

Finally, in the Table 11, the results are summarized for the 270 instances of OR-library, solved by the binarizations KMTR-BH and KMTR-Cuckoo. In addition to these results, we added Table 12. It is a comparative of different techniques that have solved the OR-library problems. We consider the results of the hyper-heuristic (CF-LAS, 2016) developed in [19], CPLEX (IBM, 2014) [48], which is a general-purpose mixed-integer programming (MIP) package used to solve linear optimisation problems. A Genetic Algorithm developed by Chu and Beasley [15], other Genetic algorithm developed by Raidl [56], and a Memetic algorihtm reported in [49].

Both k-means binarizations outperform the other methods. The average results for KMTR-BH was 0.24 and for KMTR-Cuckoo of 0.22. Regarding the mean times KMTR-BH obtained 245.6(s) and KMTR-Cuckoo 205.7(s). The configuration of the algorithms was the same as in previous executions. Every problem was executed in 30 instances.

Table 12 Detailed performance of KMTR-BH and KMTR-Cuckoo on OR-Library instances (based on average %-Gap)

Type	Reference	%-Gap
KMTR-Cuckoo	Kmeans-Transition	0.22
KMTR-BH	Kmeans-Transition	0.24
MIP	CPLEX 12.5 (IBM 2014)	0.52
GA	Raidl (1998)	0.53
GA	Chu and Beasley (1998)	0.54
Hyper-heuristic	CF-LAS (2016)	0.70
MA	Ozcan and Basaran (2009)	0.92

6 Conclusion and future work

In this article, we proposed a framework whose main function is to binarize continuous population-based metaheuristics. the performance of our framework, and the multidimensional knapsack problem was used together with the Cuckoo Search and Black Hole metaheuristics. The contribution of the different operators of the framework was evaluated, finding that the k-means transition ranking operator contributes significantly to improve the precision of the solutions. Moreover the operators Perturbation and Local Search help to improve the quality and precision of the solutions. Finally, in comparison with state of the art algorithms our framework showed a good performance.

In future works we want to investigate the behaviour of other metaheuristics in the framework. Furthermore, the framework must be verified with other NP-hard problems. Moreover, to simplify the choice of the appropriate configuration, it is important to explore adaptive techniques. From an understanding point of view of how the framework performs binarization, it is interesting to understand how the framework alters the properties of exploration and exploitation. It is also interesting to study how the velocities and positions generated by continuous metaheuristics are mapped to positions in the discrete space. Finally, we wish to explore the possibility of adapting concepts of Quantum computing to incorporate them within the framework.

Acknowledgments Broderick Crawford is supported by grant CONICYT/FONDECYT/REGULAR 1171243, Ricardo Soto is supported by Grant CONICYT /FONDECYT /REGULAR /1160455, and José García is supported by INF-PUCV 2016.

References

1. Akhlaghi M, Emami F, Nozhat N (2014) Binary tlbo algorithm assisted for designing plasmonic nano bi-pyramids-based absorption coefficient. J Mod Opt 61(13):1092–1096
2. Albo Y, Lanir J, Bak Px, Rafaeli S (2016) Off the radar Comparative evaluation of radial visualization solutions for composite indicators. IEEE Trans Vis Comput Graph 22(1):569–578
3. Alegrıa J, Túpac Y (2014) A generalized quantum-inspired evolutionary algorithm for combinatorial optimization problems. In: XXXII international conference of the Chilean computer science society SCCC, November, pp 11–15
4. Alvarez MJ, Shen Y, Giorgi FM, Lachmann A, Belinda Ding B, Hilda Ye B, Califano A (2016) Functional characterization of somatic mutations in cancer using network-based inference of protein activity. Nat Genet
5. Balas E, Zemel E (1980) An algorithm for large zero-one knapsack problems. Oper Res 28(5):1130–1154
6. Bansal JC, Deep K (2012) A modified binary particle swarm optimization for knapsack problems. Appl Math Comput 218(22):11042–11061
7. Baykasoğlu A, Ozsoydan FB (2014) An improved firefly algorithm for solving dynamic multidimensional knapsack problems. Expert Syst Appl 41(8):3712–3725
8. Beasley JE (1990) Or-library: distributing test problems by electronic mail. J Oper Res Soc 41(11):1069–1072
9. Bhattacharjee KK, Sarmah SP (2016) Modified swarm intelligence based techniques for the knapsack problem. Appl Intell, 1–22
10. Chajakis E, Guignard M (1992) A model for delivery of groceries in vehicle with multiple compartments and lagrangean approximation schemes. In: Proceedings of congreso latino ibero-americano de investigación de operaciones e ingeniería de sistemas
11. Chandrasekaran K, Simon SP (2012) Network and reliability constrained unit commitment problem using binary real coded firefly algorithm. Int J Electr Power Energy Syst 43(1):921–932
12. Changdar C, Mahapatra GS, Pal RK (2013) An ant colony optimization approach for binary knapsack problem under fuzziness. Appl Math Comput 223:243–253
13. Chen E, Li J, Liu X (2011) In search of the essential binary discrete particle swarm. Appl Soft Comput 11(3):3260–3269
14. Chih M (2015) Self-adaptive check and repair operator-based particle swarm optimization for the multidimensional knapsack problem. Appl Soft Comput 26:378–389
15. Chu PC, Beasley JE (1998) A genetic algorithm for the multidimensional knapsack problem. J Heuristics 4(1):63–86
16. Crawford B, Soto R, Cuesta R, Olivares-Suárez M, Johnson F, Olguin E (2014) Two swarm intelligence algorithms for the set covering problem. In: 2014 9th international conference on software engineering and applications (ICSOFT-EA), pp 60–69
17. Crawford B, Soto R, Olivares-Suarez M, Palma W, Paredes F, Olguin E, Norero E (2014) A binary coded firefly algorithm that solves the set covering problem. Romanian J Inf Sci Technol 17(3):252–264
18. Dey S, Bhattacharyya S, Maulik U (2015) New quantum inspired meta-heuristic techniques for multi-level colour image thresholding. Appl Soft Comput 888:999
19. Drake JH, Özcan E, Burke EK (2016) A case study of controlling crossover in a selection hyper-heuristic framework using the multidimensional knapsack problem. Evol Comput 24(1):113–141
20. Fayard D, Plateau G (1982) An algorithm for the solution of the 0–1 knapsack problem. Computing 28(3):269–287
21. García J, Crawford B, Soto R, García P (2017) A multi dynamic binary black hole algorithm applied to set covering problem. In: International conference on harmony search algorithm. Springer, pp 42–51
22. Gavish B, Pirkul H (1982) Allocation of databases and processors in a distributed computing system. Manag Distributed Data Process 31:215–231
23. Geem Z, Kim J, Loganathan GV (2001) A new heuristic optimization algorithm: harmony search. Simulation 76(2):60–68
24. Gherboudj A, Layeb A, Chikhi S (2012) Solving 0-1 knapsack problems by a discrete binary version of cuckoo search algorithm. Int J Bio-Inspired Comput 4(4):229–236
25. Gilmore PC, Gomory RE (1966) The theory and computation of knapsack functions. Oper Res 14(6):1045–1074
26. Gong T, Tuson AL (2007) Differential evolution for binary encoding. In: Soft computing in industrial applications. Springer, pp 251–262
27. Haddar B, Khemakhem M, Hanafi S, Wilbaut C (2016) A hybrid quantum particle swarm optimization for the multidimensional knapsack problem. Eng Appl Artif Intell 55:1–13
28. Hamilton R, Fuller J, Baldwin K, Vespa P, Xiao H, Bergsneider M (2016) Relative position of the third characteristic peak of the intracranial pressure pulse waveform morphology differentiates

normal-pressure hydrocephalus shunt responders and nonresponders. In: Intracranial pressure and brain monitoring XV. Springer, pp 339–345
29. Hatamlou A (2013) Black hole: A new heuristic optimization approach for data clustering. Inf Sci 222:175–184
30. Hota AR, Pat A (2010) An adaptive quantum-inspired differential evolution algorithm for 0–1 knapsack problem. In: 2010 2nd world congress on nature and biologically inspired computing (naBIC), pp 703–708
31. Ibrahim AA, Mohamed A, Shareef H, Ghoshal SP (2011) An effective power quality monitor placement method utilizing quantum-inspired particle swarm optimization. In: 2011 international conference on electrical engineering and informatics (ICEEI), pp 1–6
32. Ionita-Laza I, McCallum K, Bin X, Buxbaum JD (2016) A spectral approach integrating functional genomic annotations for coding and noncoding variants. Nat Genet 48(2):214–220
33. Kennedy JJ, Eberhart R (1997) A discrete binary version of the particle swarm algorithm. IEEE 4105:4104–4108
34. Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. Technical report, Technical report-tr06, Erciyes University, Engineering Faculty, Computer Engineering Department
35. Khalil T, Youseef H, Aziz M (2006) A binary particle swarm optimization for optimal placement and sizing of capacitor banks in radial distribution feeders with distorted substation voltages. In: Proceedings of AIML international conference, pp 137–143
36. Kong X, Gao L, Ouyang H, Li S (2015) Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm. Comput Oper Res 63:7–22
37. Kotthoff L (2014) Algorithm selection for combinatorial search problems. Surv AI Mag 35(3):48–60
38. Layeb A (2011) A novel quantum inspired cuckoo search for knapsack problems. Int J Bio-Inspired Comput 3(5):297–305
39. Layeb A (2013) A hybrid quantum inspired harmony search algorithm for 0–1 optimization problems. J Comput Appl Math 253:14–25
40. Layeb A, Boussalia SR (2012) A novel quantum inspired cuckoo search algorithm for bin packing problem. Int J Inf Technol Comput Sci (IJITCS) 4(5):58
41. Leonard BJ, Engelbrecht AP, Cleghorn CW (2015) Critical considerations on angle modulated particle swarm optimisers. Swarm Intell 9(4):291–314
42. Li X-L, Shao Z-J, Qian J-X (2002) An optimizing method based on autonomous animats: fish-swarm algorithm. Syst Eng Theory Pract 22(11):32–38
43. Liu J, Changzhi W, Cao J, Wang X, Teo KL (2016) A binary differential search algorithm for the 0–1 multidimensional knapsack problem. Appl Math Model
44. Liu W, Liu L, Cartes D (2007) Angle modulated particle swarm optimization based defensive islanding of large scale power systems. In: IEEE power engineering society conference and exposition in Africa, pp 1–8
45. Long Q, Changzhi W, Huang T, Wang X (2015) A genetic algorithm for unconstrained multi-objective optimization. Swarm Evol Comput 22:1–14
46. Martello S, Toth P (1988) A new algorithm for the 0-1 knapsack problem. Manag Sci 34(5):633–644
47. McMillan C, Plaine DR (1973) Resource allocation via 0–1 programming. Decis Sci 4:119–132
48. (2014). IBM IBM ILOG CPLEX Optimizer. http://www.ibm.com/software/commerce/optimization/cplex-optimizer/. Cited on, page 1
49. Özcan E, Başaran C (2009) A case study of memetic algorithms for constraint optimization. Soft Comput 13(8-9):871–882
50. Palit S, Sinha SN, Molla MA, Khanra A, Kule M (2011) A cryptanalytic attack on the knapsack cryptosystem using binary firefly algorithm. In: International conference on computer and communication technology (ICCCT), vol 2, pp 428–432
51. Pampara G (2012) Angle modulated population based algorithms to solve binary problems. Phd thesis, University of Pretoria, Pretoria
52. Pan W-T (2012) A new fruit fly optimization algorithm: taking the financial distress model as an example. Knowl-Based Syst 26:69–74
53. Pandiri V, Singh A (2016) Swarm intelligence approaches for multidepot salesmen problems with load balancing. Appl Intell 44(4):849–861
54. Petersen CC (1967) Computational experience with variants of the balas algorithm applied to the selection of r&d projects. Manag Sci 13(9):736–750
55. Pirkul H (1987) A heuristic solution procedure for the multiconstraint zero? One knapsack problem. Nav Res Logist 34(2):161–172
56. Raidl GR (1998) An improved genetic algorithm for the multiconstrained 0-1 knapsack problem. In: The 1998 IEEE international conference on evolutionary computation proceedings, 1998. IEEE World Congress on Computational Intelligence, pp 207–211
57. Rajalakshmi N, Padma Subramanian D, Thamizhavel K (2015) Performance enhancement of radial distributed system with distributed generators by reconfiguration using binary firefly algorithm. J Inst Eng (India): B 96(1):91–99
58. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2009) Gsa: a gravitational search algorithm. Inf Sci 179(13):2232–2248
59. Robinson D (2005) Reliability analysis of bulk power systems using swarm intelligence. In: IEEE, pp 96–102
60. Saremi S, Mirjalili S, Lewis A (2015) How important is a transfer function in discrete heuristic algorithms. Neural Comput Applic 26(3):625–640
61. Shi Y et al (2001) Particle swarm optimization: developments, applications and resources. In: Proceedings of the 2001 congress on evolutionary computation, 2001, vol 1, pp 81–86
62. Shih W (1979) A branch and bound method for the multiconstraint zero-one knapsack problem. J Oper Res Soc 30(4):369–378
63. Shuyuan Y, Min W, Licheng J (2004) A quantum particle swarm optimization. IEEE Congress Evol Comput 1:19–23
64. Simon J, Apte A, Regnier E (2016) An application of the multiple knapsack problem: The self-sufficient marine. Eur J Oper Res
65. Soto R, Crawford B, Olivares R, Barraza J, Johnson F, Paredes F (2015) A binary cuckoo search algorithm for solving the set covering problem. In: Bioinspired computation in artificial systems. Springer, pp 88–97
66. Swagatam D, Rohan M, Rupam K (2013) Multi-user detection in multi-carrier cdma wireless broadband system using a binary adaptive differential evolution algorithm. In: Proceedings of the 15th annual conference on genetic and evolutionary computation, GECCO, pp 1245–1252
67. Thaker NG et al (2016) Radar charts show value of prostate cancer treatment options. Pharmaco Econ Outcomes News 762:33–24
68. Totonchi A, Reza M (2008) Magnetic optimization algorithms, a new synthesis. In: IEEE international conference on evolutionary computations
69. Wang I, Zhang Y, Zhou Y (2008) Discrete quantum-behaved particle swarm optimization based on estimation of distribution for combinatorial optimization. In: IEEE evolutionary computation, pp 897–904
70. Wang L, Zheng X-L, Wang S-Y (2013) A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem. Knowl-Based Syst 48:17–23

71. Weingartner HM (1963) Mathematical programming and the analysis of capital budgeting problems. Markham Pub. Co.
72. Weingartner MH, Ness DN (1967) Methods for the solution of the multidimensional 0/1 knapsack problem. Oper Res 15(1):83–103
73. Yang X-S (2009) Firefly algorithms for multimodal optimization. In: International symposium on stochastic algorithms. Springer, pp 169–178
74. Yang X-S (2010) A new metaheuristic bat-inspired algorithm. In: Nature inspired cooperative strategies for optimization (NICSO 2010). Springer, pp 65–74
75. Yang X-S, Deb S (2009) Cuckoo search via lévy flights. In: World congress on nature & biologically inspired computing, 2009. naBIC 2009, pp 210–214
76. Yang Y, Yi M, 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
77. Zakaria D, Chaker D (2015) Binary bat algorithm: on the efficiency of mapping functions when handling binary problems using continuous-variable-based metaheuristics. Comput Sci Appl 456:3–14
78. Zhang B, Pan Q-K, Zhang X-L, Duan P-Y (2015) An effective hybrid harmony search-based algorithm for solving multidimensional knapsack problems. Appl Soft Comput 29: 288–297
79. Zhang G (2011) Quantum-inspired evolutionary algorithms: a survey and empirical study. J Heuristics 17(3):303–351
80. Zhang X, Changzhi W, 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
81. Zhao J, Sun J, Wenbo X (2005) A binary quantum-behaved particle swarm optimization algorithm with cooperative approach. Int J Comput Sci 10(2):112–118
82. Zhifeng W, Houkuan H, Xiang11 Z (2008) A binary-encoding differential evolution algorithm for agent coalition. J Comput Res Dev 5:019
83. Zhou Y, Bao Z, Luo Q, Zhang S (2016) A complex-valued encoding wind driven optimization for the 0-1 knapsack problem. Appl Intell, 1–19
84. Zhou Y, Chen X, Zhou G (2016) An improved monkey algorithm for a 0-1 knapsack problem. Appl Soft Comput 38:817–830