Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Modified Artificial Bee Colony Algorithm for Multiple-Choice Multidimensional Knapsack Problem

ARIJ MKAOUAR[1], SKANDER HTIOUECH[2,3], AND HABIB CHABCHOUB[4]
[1]OLID Laboratory, ISGIS, University of Sfax, BP 1164, 3021, Sfax, Tunisia
[2]CS and AI department, University of Jeddah, 21959, Jeddah, Saudi Arabia
[3]DES Research Unit, Faculty of Sciences of Sfax, University of Sfax, BP 1171-3000, Sfax, Tunisia
[4]College of Business, Al Ain University, P.O.Box: 112612, Abu Dhabi, UAE

Corresponding author: Arij Mkaouar (arij.mkaouar@yahoo.fr).

ABSTRACT The multiple-choice multidimensional knapsack problem (MMKP) is a well-known NP-hard problem that has many real-time applications. However, owing to its complexity, finding computationally efficient solutions for the MMKP remains a challenging task. In this study, we propose a Modified Artificial Bee Colony algorithm (MABC) to solve the MMKP. The MABC employs surrogate relaxation, Hamming distance, and a tabu list to enhance the local search process and exploit neighborhood information. We evaluated the performance of the MABC on standard benchmark instances and compared it with several state-of-the-art algorithms, including RLS, ALMMKP, ACO, PEGF-PERC, TIKS-TIKS^2 and D-QPSO. The experimental results reveal that MABC produces highly competitive solutions in terms of the best solutions found, achieving approximately 2% of the optimal solutions with trivial (milliseconds) CPU time. The Kruskal-Wallis test revealed that there was no statistically significant difference in the objective function values between the MABC algorithm and other state-of-the-art algorithms (H = 0.31506, p = 0.98882). However, for CPU efficiency, the test showed a statistically significant difference (H = 84.90850, p = 0), indicating that the MABC algorithm exhibited significantly better CPU efficiency (with shorter execution times) than the other algorithms did. Along with these findings, the ease of implementation of the algorithm and the small number of control parameters make our approach highly adaptive for large-scale real-time systems.

INDEX TERMS Artificial bee colony algorithm, multiple-choice multidimensional knapsack problem, hamming distance, surrogate relaxation.

I. INTRODUCTION

The MMKP is a generalization of the classical knapsack problem (KP)[1]. It is significant because it can model a wide range of real-time applications such as resource allocation[2], intelligent transportation systems[3], logistics[4], quality of service (QoS)[5], [6], web service composition[7], Energy-Efficient Offloading in Mobile Edge Computing[8], medicine[9], budgeting problems[10], hardware design[11], and cloud computing[12].

Formally, the MMKP can be expressed as a set of items divided into n disjoint groups and an m-dimensional resource constraint represented by the vector of resource availability $b^k = \{b^1, \dots, b^m\}$. Item j in group i has a non-negative profit value v_{ij} and consumes a certain amount r_{ij}^k of resource k (k = 1, …, m). The decision variable x_{ij} = 1 if item j from class i is selected and x_{ij} = 0 otherwise. The MMKP is then defined as

$$Maximize \sum_{i=1}^{n}\sum_{j=1}^{n_i} v_{ij}x_{ij} \quad (1)$$

$$s.t. \sum_{i=1}^{n}\sum_{j=1}^{n_i} r_{ij}^k x_{ij} \le b^k, k = 1, \dots, m \quad (2)$$

$$\sum_{j=1}^{n_i} x_{ij} = 1, i = 1, \dots, n \quad (3)$$

$$x_{ij} \in \{0,1\}, i = 1, \dots, n, j = 1, \dots, n_i \quad (4)$$

As a variant of the classical KP, the MMKP can be considered as a combination of two challenging problems: the multidimensional knapsack problem (MKP)[13] and the multi-choice knapsack problem (MCKP)[14]. It involves a linear objective function under two types of linear constraints: (2) and (3). The first is a multidimensional constraint and the second is a choice constraint. If constraint (3), which limits each group to a single item, is relaxed, the problem is reduced to MKP[15]. Constraint (2) guarantees that the knapsack capacities are respected, and if the resource constraint is relaxed to a single dimension (m = 1), the problem is reduced to an MCKP[16].

The MMKP has a potentially wide range of practical applications. However, developing an efficient and effective algorithm for the problem is challenging because it is an NP-hard problem[17] as it is not trivial to find even a feasible solution within polynomial runtime complexity, particularly for large-scale problems.

NP-hard problems have numerous local minima that can be challenging to escape. This is further aggravated by the cycling phenomenon, wherein the algorithm repeatedly visits a candidate solution, thereby wasting time and causing it to fall into a local optimum.

Recently, metaheuristics based on nature-inspired algorithms have gained attention as a solution to the MMKP and other complex optimization problems. These algorithms are inspired by the behavior and decision-making processes of natural systems, such as the genetic algorithm (GA)[18], particle-swarm optimization (PSO)[19], ant colony optimization (ACO)[20], [21], biogeography-based optimization (BBO)[22], harmony search (HS)[23], and artificial bee colony (ABC) algorithm[24]. By combining metaheuristics with mathematical optimization techniques, they can effectively solve complex optimization problems and produce near-optimal solutions.

The use of metaheuristics based on nature-inspired algorithms in practical problems highlights the importance and potential of these algorithms for real-world applications. These algorithms can effectively and efficiently solve complex optimization problems, making them useful in various fields[25], [26].

Since its invention by Karaboga[24], the ABC algorithm has received increasing attention owing to its flexibility, simplicity of employment, and small number of control parameters[27]. Compared to other evolutionary algorithms, the ABC algorithm can escape local optima[28], [29] in several real-world problems[30]–[32] and is widely used in the field of combinatorial optimization problems[33] such as traveling salesman[34], vehicle routing[35], [36], graph coloring[37], team orienteering[38], bioinformatics[39], web service composition[40], social network analysis[41], timetabling[42]–[44], controller design[45], and image processing[46].

The ABC algorithm contains three main phases: employed-bee, onlooker-bee, and scout-bee. The employed-bee and onlooker-bee phases are dedicated to the exploitation of the search space, whereas the scout-bee phase is dedicated to the exploration of the search space. The exploration strategy of the ABC algorithm, which is based on a stochastic pattern search process, delivers excellent performance. However, similar to other evolutionary algorithms, it encounters performance challenges during the exploitation process[47].

The deficiencies in the ABC exploitation process are caused by several factors. In the employed-bee phase, the local search process is related to neighborhood information, which limits the efficiency of the exploitation process owing to the restricted information that a neighborhood can offer. A similar drawback affects the onlooker-bee, because the same structure is applied in the onlooker-bee phase. Furthermore, the fitness structure used in ABC maintains only food sources with high amounts of nectar; however, low-nectar solutions may also contain useful information. These factors lead to an imbalance between exploitation and exploration in the search process, causing delayed convergence and falling into a local optima[48].

In this study, we propose a Modified Artificial Bee Colony (MABC) algorithm to solve the MMKP based on the ABC algorithm. The MABC algorithm improves the performance of the ABC algorithm by integrating three distinct techniques, namely, surrogate relaxation, Hamming distance, and tabu list, to handle the combinatorial nature of the problem, increase population diversity, and facilitate faster convergence to find near-optimal solutions within a short computational time. The MABC algorithm adopts the Hamming distance to measure the dissimilarity between candidate solutions, which is defined as the number of positions (groups) at which two solutions differ. The algorithm employs a stochastic selection process to generate new solutions within a predefined Hamming distance from the current solutions. This approach improves the population diversity, thereby avoiding convergence to local optima. Furthermore, the MABC algorithm employs a surrogate relaxation approach to address the combinatorial nature of the MMKP problem. By combining surrogate relaxation with Hamming distance techniques, the MABC algorithm aims to accelerate the convergence and find near-optimal solutions in reduced computational time. Finally, the MABC algorithm integrates a tabu list that monitors the recently visited solutions, thereby preventing the algorithm from revisiting them. This technique enhances the exploration of the solution space and assists in avoiding the local optima.

The approach was validated on standard benchmark problem instances and compared with several state-of-the-art algorithms in literature.

The remainder of this paper is organized as follows. Section 2 presents a brief review of the relevant literature.

We describe our MABC algorithm in Section 3. Section 4 presents and discusses the extensive computational results obtained for the known benchmark problems. Finally, Section 5 concludes the study.

II. LITERATURE REVIEW

MMKP is strongly constrained and NP-hard[49]. Consequently, the search space grows exponentially with the problem size[17], which renders scanning highly difficult or even impossible in practice, despite advancements in computer technologies. In particular, it is challenging to identify a good solution quality without falling into a local optimum when scanning most of the search space.

Several exact approaches have been proposed to solve the MMKP problem[50]–[52] most of which use a branch-and-bound algorithm. Khan[52] suggested a combination of a branch-and-bound algorithm and linear programming. Sbihi[51] described an exact branch-and-bound algorithm that explores the search tree using the best-first strategy. In this approach, the upper bounds of the objective function are computed by reducing multiple dimensions to one and transforming the problem into an MCKP problem. Hence, the computational results reported in Sbihi[51] indicate that the algorithm outperformed Khan's approach[52]. Razzazi and Ghasemi[53] used a more powerful branch-and-bound scheme based on a depth-first strategy to explore the search tree. They calculated the upper bounds using the surrogate relaxation of the problem. Their algorithm provides better results than those of Sbihi[51]. Ghasemi and Razzazi[50] developed an exact algorithm based on an approximate core to solve MMKP. They obtained promising results with up to five knapsack constraints and 1000 items.

Nevertheless, exact approaches can only deal with problems of a limited size ($n = 100$ and $m = 10$)[54]. For real-time decisions, exact algorithms are not feasible owing to their complexity and requirement of a fast system response. Therefore, approximation algorithms are viable options for solving the MMKP, particularly in cases where a precise optimal solution is not required and computational time is a significant constraint.

For larger instances, several heuristics have been proposed to determine near-optimal solutions within acceptable computation time[54]. In 1997, Moser *et al.*[55] introduced the first heuristic algorithm for resolving the MMKP based on Lagrangian relaxation and repetitive permutation. The approach was subsequently improved by Akbar *et al.*[56].

Further, Hifi *et al.*[57] used a reactive local search (RLS) and a modified reactive local search (MRLS), which yielded better results than those of Moser *et al.*[55]. Cherfi and Hifi [58] proposed a hybrid algorithm combining local branching with column generation, which outperformed all previous approaches.

Combinations of linear programming relaxation and other techniques are often used in MMKP algorithms to solve the reduced problem[59]–[61]. Cherfi[62] extended the approach proposed by Cherfi and Hifi[58] to improve the quality of solutions by combining column generation techniques and local search. Ren and Feng[63] presented an ACO approach following the scheme of a max-min ant system to solve the MMKP problem. Crévits *et al.*[64] introduced a semi-continuous relaxation approach to solve the MMKP. In their approach, relaxation is used at each iteration to generate an upper bound and then create a sub-problem that can be solved to find a lower bound. Pseudo cuts are also produced to prevent falls into the local optima.

Mansi *et al.*[60] described another hybrid approach based on iterative relaxation that applies new cuts to generate a reduced problem and a reformulation procedure. Additionally, Htiouech *et al.*[65] used a surrogate constraint combined with an oscillation method to solve the MMKP. Subsequently, Htiouech and Alzaidi[66] divided the MMKP into small sub-problems and used an agent-based approach to solve the reduced problem. Xia *et al.*[67] proposed a first-level tabu search algorithm. Their proposed algorithm performs fairly well compared with legacy heuristic approaches.

Gao *et al.*[61] described a new iterative pseudo-gap enumeration based on a new family of pseudo-cuts resulting from the reduced cost constraint of non-basic variables. Dong *et al.*[68] proposed an enhanced quantum particle swarm optimization algorithm for MMKP that prioritizes effective genes and reserves particles with greater revolutionary potential. The algorithm employs a mutation based on elite genes to prevent local optimization when the population diversity decreases.

Caserta *et al.*[69] defined a primary mathematical model for solving the MMKP. Their model addresses complex system reliability and uses a new robust formulation characterized by second-order cone programs. In this model, the resource consumption values of items are nondeterministic. The authors demonstrated the ability to convert a nondeterministic MMKP into an integer linear program without extra complexity. Mkaouar *et al.*[70] developed an algorithm that uses the ABC algorithm to resolve the MMKP. Their proposed algorithm, inspired by the general behavior of the honeybee swarm, provided better quality solutions for medium and large scale instances compared to other reported approaches.

Mansini and Zanotti[71] proposed a new approach for solving this problem. The method solves sub-problems of increasing size using a recursive variable fixing process until an optimality condition is satisfied. Syarif *et al.*[72] analyzed three different GA and evaluated the performance of several heuristic algorithm approaches to solve the MMKP.

Yang *et al.*[73] applied a memetic algorithm to the MMKP. The authors designed a repair heuristic based on a tendency function with human experience through experiments using genetic algorithms. Lamanna *et al.*[74] provided a new variant of the heuristic framework kernel search applied to the MMKP. Dellinger *et al.*[75] proposed simple strategies that generate bounded solutions for the MMKP.

Despite the existence of several exact and approximate heuristic algorithms for solving the MMKP, new algorithms must be developed. This is for several reasons, including the fact that the MMKP is an NP-hard problem, meaning that obtaining exact solutions for large problem instances is a difficult task. Moreover, many existing algorithms are computationally intensive and require considerable computational time to provide the best possible solutions, particularly for large problems. The computational time required to solve the MMKP problem can be reduced by developing more efficient algorithms. Finally, new algorithms may need to be developed to handle new constraints, objectives, or uncertainties or to adapt to new problem instances or settings, which may require more flexible and adaptable solutions.

Despite the diversity of methods used in research on the MMKP, none of the methods have leveraged dissimilarity and similarity measures between solutions. Such a measure can be highly valuable because it provides numeric values quantifying the relative positions (distances) of solutions with respect to each other in the search space. Therefore, this concept can provide significant flexibility for the algorithm to jump from one current local search area to another, and consequently explore several different zones of the search space.

III. MABC

To the best of our knowledge, the ABC algorithm has not yet been used for the MMKP, except in the study by Mkaouar *et al.* [70] that presents an algorithm inspired by the general behavior of a honeybee swarm. However, this study represents the different phases of the ABC algorithm for the MMKP.

The ABC algorithm was first proposed by Karaboga[24] to solve continuous and discrete problems. Subsequently, Karaboga and Basturk[28] compared the performance of the ABC algorithm to that of the GA, differential evolution (DE), PSO, and evolutionary algorithm (EA), and tested them using five multidimensional numerical test problems. The experimental results show that ABC escaped falling into a local minimum, was more proficient for multivariable and multimodal function optimization, and outperformed DE, PSO, and EA[28], [29].

The ABC algorithm simulates the intelligent comportment of a honeybee swarm while probing for a food source (solution). In an ABC, a potential solution for the considered problem is a symbol of a food source, and the quality of this solution depicts the quantity of nectar in this food source. ABC adopts a colony model divided into three categories according to the task performance.

Employed-bees: Each employed bee is a distinct food source. Employed-bees are responsible for investigating nectar food sources in areas or neighborhoods already visited by them. An employed-bee modifies the food source (current solution) in its memory depending on the local information (visual information) and assessments of the nectar amount (fitness value) of the new source (new solution). If the quantity of nectar in the new solution is higher than that in the current solution, the bee memorizes the new position and abandons the old one. Otherwise, it retains the position of the solution in memory. Moreover, it shares information about food sources with a certain probability with the bees residing in the hive (onlooker-bees).

Onlooker-bees: receive information from employed-bees and evaluate the quality of the food source. Similar to the employed-bees, onlooker-bees attempt to improve the solutions using a greedy search strategy.

Scout-bees: The employed-bee and onlooker-bee phases are dedicated to the exploitation of the search space, whereas the scout-bee phase is dedicated to the exploration of the search space. Scouts search for new food sources in new areas. An employed-bee becomes a scout-bee when the quality of a food source does not improve after a predetermined number of attempts, called the “limit.”

This cycle (employed-bee, onlooker-bee, and scout-bee phases) is repeated until “maxCycle” (maximum number of cycles) is reached. Subsequently, the best global solution is returned by the algorithm.

The main steps of the ABC algorithm are summarized in Algorithm 1.

Algorithm 1: MAIN STEPS OF ABC ALGORITHM

```
ABC algorithm
Output: Global best solution found
1. Initialization
Repeat
2. Employed-bee phase
3. Onlooker-bee phase
4. Scout-bee phase
5. Refresh memory
Until (one of the stop conditions is satisfied)
```

The different phases of the ABC algorithm were modified to improve its performance. The following subsections describe each step in more details.

A. Initialization of population

The generation of an initial population in an optimization metaheuristic is important because it affects the search in future iterations and significantly influences the final solution. The random method aims to generate random solutions to produce greater diversity, which is an important factor for determining the quality of the final solution. The random greedy method may generate a population with good fitness solutions; however, there is a risk of rapid convergence toward a local optimum[76].

Because it may not produce feasible solutions for the MMKP within a reasonable computation time, the random initialization method for the candidate solutions of the initial population used in the standard ABC is not suitable for the MMKP. Therefore, in this study, a new initialization method based on the Hamming distance (see Section V.B.1) was applied to create the initial population (initializeSolution procedure) to increase the diversity of the population and help avoid getting stuck in local optima.

The principle of this procedure is to assign the n groups of the problem to N classes with $N < n$ and $n \bmod N = 0$. The groups are randomly assigned to classes with the same number of groups in each class. The problem here is reduced to N sub-problems (classes) containing each of the $n_N = n/N$ groups. Each reduced problem P' is individually solved. The combination of N partial solutions for the N sub-problems can be a candidate for a complete feasible solution for the MMKP. The initializeSolution procedure is repeated for each candidate solution X_l of the initial population with l = {1, ..., SN}, and SN is the size of the population.

The main steps for the solution generation are presented in Algorithm 2.

Algorithm 2: INITIALIZESOLUTION PROCEDURE

```
initializeSolution procedure
Input: index $l$ of the candidate solution $X_l$ with $l$ = {1, ..., SN}
Output: solution $X_l$
1.  /* initialization phase*/
2.  $X_l \leftarrow$ Null
3.  $\Theta = \{1, ..., n\}$ // Θ is the set of remaining groups not selected
4.  $\Theta' = \emptyset$ // the set of groups already selected
5.  Limit the bound on the availability of resources $k$ ($k = 1, ..., m$) to
    $b_p^k = \frac{b^k}{N}$ for each subset
6.  dh= constant // number of items to be exchanged from a candidate
    solution
7.  While ($\Theta' \neq \emptyset$) // while there is a group not yet initialized
8.          Select $\Theta''$// $\Theta''$ is a randomly selected $n_N = n/N$ groups
9.          $\Theta \leftarrow \Theta \setminus \Theta''$ // $\Theta$ = the remaining groups after the difference
    between $\Theta$ and $\Theta''$
10.         $\Theta' \leftarrow \Theta' \cup \Theta''$ // $\Theta'$= the union of $\Theta'$ and $\Theta''$
11.         /*Solve sub-problem $\Theta''$ */
        Set $x_{ij}$ =1 with $j$ the item selected for group $i \in \Theta''$ having the
lowest resource consumption, $min \sum_{j=1}^{n_i} \frac{r_{ij}^k}{b^k}$, $i \in \{1, ..., n\}$
12.         /* $d$ random exchanges between items*/
13.         $d = 1$
14.         While ($d$ < dh)
15.               Select group $i$ randomly, $i \in \{1, .., n\}$
16.               Select item $j'$ randomly, $j' \in \{1, .., n_i\}$, with $x_{ij} = 0$ and
    $j \neq j'$
17.               Check if the resource constraints are satisfied
18.               Boolean violatedres ← false
19.               For int $k$ = 0 to $m$
20.                    Res = 0
21.                    For each group $i \in \Theta''$
22.                         Res = Res + $r_{ij}^k$
23.                    End For
24.                    If (Res $- r_{ij}^k + r_{ij'}^k > b_p^k$)
25.                       Violatedres ← true // the resource constraints are
    violated
26.                       Break // quit loop; For int k = 0 to m
27.                   End If
28.               End For
29.               If (violatedres = false) // no resource constraints
    violated
30.               $x_{ij} \leftarrow 0$
31.               $x_{ij'} \leftarrow 1$
32.               Update the $X_l$
33.           End If
34.               $d = d+1$
35.        End While
36. Return $X_l$
```

B. Modified employed-bee phase

This phase aims to enhance the performance of the ABC algorithm by obtaining superior quality solutions while avoiding convergence to local optima[47], [48], [77].

1) DISTANCE MEASURE: HAMMING DISTANCE

In this study, we incorporated the concept of a distance measure to enhance the exploitation search process. Manipulating the distances between solutions facilitates the localization of relative positions of the solutions. It also offers flexibility of movement within the search space from the position of the current solution to that of another by acting on a predefined part of the current solution. The size of the manipulated part represents the distance between the current and newly generated solution. The distance between solutions reflects the degree of similarity. If the distance between the two solutions is small, then the solutions are similar and located within a neighborhood search area, whereas if the distance between the two solutions is large, then the solutions are dissimilar and localized in different search areas.

The MABC algorithm utilizes the Hamming distance as a distance metric because of its simplicity and effectiveness in capturing the differences between two solutions based on the number of different bits. Specifically, in the context of the MMKP, the Hamming distance is adept at capturing the differences between two solutions based on the number of groups in which the corresponding items differ.

In information theory, the Hamming distance between two binary strings a and b is measured by performing the XOR operation ($a \oplus b$) and then counting the total number of ones in the resultant string[78]. Typically, the Hamming distance between two vectors a and b of the same length n is given by

$$\mathrm{dh}(a,b) = \sum_{i=1}^{n} (a_i \oplus b_i) \qquad (5)$$

The solution for the MMKP is represented as a vector X of length n (number of groups). X is used to indicate item j with $j \in \{1, .., n_i\}$, which is selected from each group i with $i \in \{1, .., n\}$. Fig. 1 illustrates the structure of solution X for an instance containing ten groups (n = 10) and five items (n_i = 5).

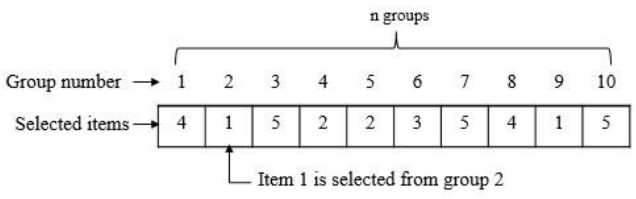

FIGURE 1. Structure of a solution X for the multiple-choice multidimensional knapsack problem (MMKP).

In our approach, we define the distance between two solutions as the number of groups in which selected items differ. The distance d between two solutions X and X' of length n is given by

$$d(X, X') = \sum_{i=1}^{n} |X[i] - X'[i]| \qquad (6)$$

$$with \begin{cases} |X[i] - X'[i]| = 0 \; if \; X[i] = X'[i] \\ |X[i] - X'[i]| = 1 \; if \; X[i] \neq X'[i] \end{cases}$$

An example of the calculation of the Hamming distance between the two MMKP candidate solutions is shown in Fig. 2.

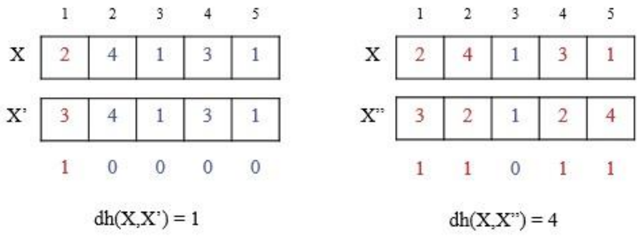

FIGURE 2. Illustration of hamming distance calculation: X→X' = 1 step; X→X''= 4 steps.

The unit of the Hamming distance measure (dh) is referred to as a step (the number of groups in which the corresponding items differ). For example, if dh (s, s') = 2, s is two steps away from s'.

This approach aims to manipulate the search depth in the exploitation phase by expanding (selecting large steps between solutions) or narrowing (selecting small steps between solutions) the local area around the solution (Fig. 3). This facilitates the exchange of a set of items in a single iteration, and consequently helps the algorithm to quickly converge to promising regions of the solution space. Specifically, the proposed algorithm generates a candidate solution s' in the neighborhood of the current solution s with dh (s, s') = d steps (number of items to exchange) from the current solution s by performing d exchanges E (i, j, h) of d items satisfying x_{ih} = 1, with d new items belonging to the same group satisfying x_{ij} = 0.

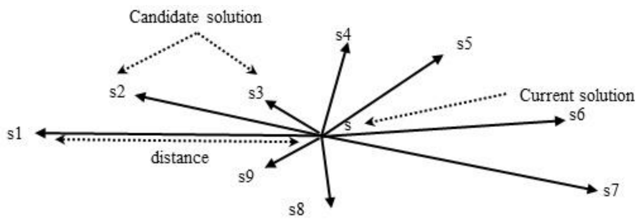

FIGURE 3. Exploitation process with hamming distance.

Algorithm 3 presents the main steps for generating a new solution V from the current solution X with dh (X, V) = d.

Algorithm 3: HAMMING DISTANCE ALGORITHM

```
Hamming distance algorithm
Input: current solution X
Output: new candidate solution V
1. dh = constant
2. For d = 1 to dh
3.     select item h with $x_{ih}$ = 1 from X
4.     select item j with $x_{ij}$ = 0 // j and h belong to the same group
5.     replace item h with j // a mutant solution V is generated from X
6.     if the new solution V is feasible, then replace V with X
7. End For
8. Return V
```

2) SURROGATE RELAXATION

Each solution has more than one neighboring candidate solution. The choice of movement depends only on the information from the neighborhood domain of the current solution. Therefore, it is important to define a neighborhood relation between the solutions in the search space. In this study, the neighborhood relation was based on the surrogate constraint relaxation information. The surrogate relaxation technique is mainly used to prevent the algorithm from getting stuck in local optima. This involves approximating the original optimization problem with a simpler, relaxed problem that is easier to solve. By relaxing the problem, the algorithm can explore a larger solution space and avoid becoming trapped in local optima. Glover[79] proposed a surrogate constraint obtained by substituting the constraints of a problem with a single constraint to obtain approximate near-optimal solutions for integer programming problems. Surrogate relaxation has been proven to be efficient in the exploitation of several knapsack problems[65], [80].

Htiouech *et al.*[65] used surrogate relaxation information according to the search direction: add move, drop move (infeasible solution case), or swap move (feasible solution case). Here, we are interested only in the swap move strategy, because our algorithm only deals with feasible solutions. The surrogate relaxation algorithm (swap move) proposed by Htiouech *et al.* continues through all groups and items in each group, and performs intensive mathematical tests and computations. Exchanges between items are performed whenever an improvement is detected. The algorithm stops only when no further improvement is possible, which making it computationally heavy.

However, in our study, the computational time was considered a critical measure of the quality of the obtained solutions. Consequently, adjustments were made to the surrogate relaxation method used by Htiouech *et al.*[65] to overcome its limitations in terms of time complexity. Therefore, in the employed-bee phase of the MABC, we combine the Hamming distance (local search depth control) and the surrogate relaxation information structure to utilize its solution improvement efficiency without increasing the time complexity of the algorithm. Thus, only dh exchanges are permitted in the current solution. The choice rule for exchanges between items is based on the information provided by the surrogate relaxation.

During this step, d_{emp} exchanges are performed for each candidate solution (Steps 18–21 in Algorithm 4). In each exchange *E (i, j, h)*, item *h* is selected from group *i* and replaced by item *j* belonging to the same group. The choice of the d_{emp} groups (items from the current solution to be exchanged) is based on the lowest ratio utility (Step 19 in Algorithm 4) given by

$$u_{ij} = \frac{v_{ij}}{\sum_{k=1}^{m} \frac{r_{ij}^k}{b^k}} \Bigg| x_{ij} = 1, i = 1,..,n, j = 1,..,n_i \quad (10)$$

For each selected item with $x_{ih} = 1$, surrogate relaxation is used (Step 20 in Algorithm 4) to select a new item $x_{ij} = 0$ from the same group to maximize (11).

$$maximize \left\{ \frac{v_{ij}/s_{ij}}{v_{ih}/s_{ih}} \Bigg| x_{ij} = 0, x_{ih} = 1 \right\} \quad (11)$$

$$s_{ij} = \sum_{k=1}^{m} \frac{A_k}{(\Delta^k)^2} r_{ij}^k$$

$$with\ \Delta^k = b^k - \sum_{i=1}^{n} \sum_{j=1}^{n_i} r_{ij}^k x_{ij} \big| x_{ij} = 1, k = 1,..,m \quad (12)$$

$$and\ A_k = \sum_{i=1}^{n} \sum_{j=1}^{n_i} r_{ij}^k \big| x_{ij} = 0$$

In (11), the term v_{ij} / v_{ih} represents the gain of the profit value, and s_{ij} / s_{ih} represents the gain in terms of the remaining available resources. Consequently, the choice of the new item *j* ($x_{ij} = 0$) to be exchanged instead of the old item *h* ($x_{ih} = 1$) is made to improve the quality of the solution in terms of the maximization of (i) profit and (ii) savings of aggregate resources.

After the exchange, the decision variable x_{ij} changes from 0 to 1, and x_{ih} changes from 1 to 0. Therefore, a new solution V_l is generated (Step 22 in Algorithm 4) from the current solution X_l (with *l* in {1, ..., SN}), where dh $(X_l, V_l) = d_{emp}$. A greedy selection based on the value of the objective function is performed between the newly generated solution and the current solution (Steps 23–29 in Algorithm 4).

C. Modified onlooker-bee phase

The probability of selecting a candidate solution in this phase (Steps 32–34 in Algorithm 4) is inspired by the Gbest-guided artificial bee colony (GABC) algorithm[81], which is given by

$$\text{prob}_i = \left(\frac{0.9 * \text{fitness}\ (X_i)}{\text{fitness}_{\text{best}}} \right) + 0.1 \quad (13)$$

where fitness (X_i) is the fitness value of the current solution and fitness$_{\text{best}}$ is the highest fitness value.

A feasible candidate solution with relatively low nectar content may yield a better global solution than another candidate solution with higher nectar content. Therefore, the greedy selection between the current solution X_l and the new solution V_l is based on the highest fitness value computed by (14) instead of (9).

$$\text{fitness}_l(X_l) = \frac{b^k}{\sum_{i=1}^{n} \sum_{j=1}^{n_i} r_{ij}^k}, k = 1,..,m, \forall\ l \in \{1,..,\text{SN}\} \quad (14)$$

The depth of the search is controlled using the Hamming distance (dh). In other words, dh = d_{on} maximum number of exchanges is allowed (Steps 39–42 in Algorithm 4) to obtain a new candidate solution V_l from the current solution X_l.

$$\text{dh}\ (X_l, V_l) = d_{on}, l \in \{1,..,\text{SN}\} \quad (15)$$

For each candidate solution, d_{on} items ($x_{ij} = 1$) with the lowest profit selected using (16) (Step 40 in Algorithm 4) are exchanged randomly with the new item ($x_{ij} = 0$), thereby satisfying the choice constraint (Step 41 in Algorithm 4). No exchange is allowed, unless this leads to a feasible solution.

$$\min_{\substack{1 \le i \le n \\ 1 \le j \le n_i}} v_{ij} x_{ij} \big| x_{ij} = 1 \quad (16)$$

After d_{on} exchanges, a greedy selection between the new solution V_l and the current solution X_l is performed, and only the solution with the highest fitness value (14) is retained (Steps 43–50 in Algorithm 4). Equation (14) (the bound of the resources availability per consumed resources) ensures an improvement in the quality of the retained solution (neither the current solution X_l nor the newly generated solution V_l) in terms maximizing the remaining available resources, which reduces the overall load on the resources during the subsequent iterations of the exploitation process.

D. Modified scout-bee phase

The employed-bee and onlooker-bee phases could be considered perturbation processes that aim to change the trajectory of the search to release the solution when it is stuck in a potential local optimum, and further improve the solution without losing the progress made by the exploitation process. If the solution cannot be further improved, it is completely re-initialized with a randomly generated new solution (scout bee) using the initializeSolution procedure (Algorithm 2) (Steps 59–64 in Algorithm 4).

A tabu list (TL) is used in this study. This involves tracking recently explored solutions and temporarily excluding them from the search. This helps the algorithm avoid revisiting the same solutions repeatedly and efficiently exploring new areas of the solution space, thus accelerating convergence. TL is updated using the first-in-first-out (FIFO) strategy. Consider V_l as the new solution to be inserted into TL, and X_l as the oldest solution in the list. The TL vector is updated as follows

$$\mathrm{TL} = \mathrm{TL} - X_l + V_l \tag{17}$$

E. General approach

Algorithm 4 presents the main steps of the MABC. The algorithm begins by generating the initial population (Steps 11–15 in Algorithm 4), followed by the main loop containing the employed-bee, onlooker-bee, and scout-bee phases of the algorithm. The employed-bee (Steps 16–30 in Algorithm 4) and onlooker-bee (Steps 31–57 in Algorithm 4) phases aim to improve the quality of solutions. In the employed-bee phase, if the generated solution achieves a better objective value than the current solution, the current solution is updated. However, in the onlooker-bee phase, the solution with the best remaining resource value is retained. If a solution reaches the maximum for non-improvement (limit), the solution is completely reset by the scout bee (Steps 59–64 in Algorithm 4). The three main steps of the algorithm (employed-bee, onlooker-bee, and scout-bee phases) are repeated until the maximum number of cycles is reached (Step 66 in Algorithm 4). The global best solution is memorized in each cycle (Step 65 of the algorithm) and then returned to the end of the algorithm (Step 67 in Algorithm 4).

The pseudo code for the proposed approach is presented in Algorithm 4.

Algorithm 4: HIGH-LEVEL PSEUDO CODE FOR MABC

```
MABC
Output: BGS
1.  /*Parameter initialization*/
2.  predefine limit
3.  predefine maxCycle
4.  predefine $d_{emp}$ // $d_{emp}$=dh ($X_l$, $V_l$) is the number of exchanges used in the employed-bee phase
5.  predefine $d_{on}$ // $d_{on}$=dh ($X_l$, $V_l$) is the number of exchanges used in the onlooker-bee phase
6.  TL length= SN // the TL has the same tenure as the population size
7.  For $l$ = 1 to SN
8.        trial$_l$ ← 0
9.  End For
10. /*Generate an initial population*/
11. For $l$ ← 1 to SN
12. Use the initializeSolution procedure (algorithm 2) to generate a candidate solution $X_l$
13.           TL$_l$ ←$X_l$ // initially, the TL contains the initial population
14. End For
15. Cycle ← 1
16. /* Employed-bee phase */
17. For $l$ = 1 to SN
18.       For $i$ = 1 to $d_{emp}$
19.           Select $x_{ih}$ from $X_l$ having the lowest $u_{ij}$ computed using (10)
20.           Select $x_{ij}$ that maximizes (11) // the choice of item j is based on the surrogate relaxation
21.       End For
22. Calculate the fitness value fitness ($V_l$) // $V_l$ is generated from $X_l$ after $d_{emp}$ exchanges
23.    If $f$ ($V_l$) > $f$ ($X_l$) and $V_l \notin$ TL // f is the objective function value of the considered solution
24.        $X_l$ ← $V_l$
25.        Update TL using (17)
26.        trial$_l$ ← 0
27.    Else
28.        trial$_l$ ← trial$_l$ + 1
29.     End If
30. End For
31. /*Onlooker-bee phase*/
32. For $l$ = 1 to SN
33.      Calculate the probability values prob$_l$ using (13)
34. End For
35. $t$ ← 1
36. $l$ ← 1 // set index l of the current solution at 1 with $l \in \{1, ..., \mathrm{SN}\}$
37. While ($t$ < SN) do
38.    If rand (0,1) < prob$_l$ // rand (0,1) returns a real value between 0 and 1
39.    For $i$ = 1 to $d_{on}$
40.         Select $x_{ih}$ using (16) with $x_{ih}$ = 1 from $X_l$
41.         $x_{ij}$ ← rand ($n_i$) with $x_{ij}$ = 0 and $j \neq h$ // rand ($n_i$) returns an integer between 1 and $n_i$
42.    End For
43.         Calculate fitness ($V_l$) using (14) // $V_l$ is the mutant of $X_l$ after performing $d_{on}$ exchanges
44.        If fitness ($V_l$) > fitness ($X_l$) and $V_l \notin TL$
45.            $X_l$ ← $V_l$
46.            trial$_l$ ← 0
47.            Update TL using (17)
48.        Else
49.            trial$_l$ ← trail$_l$ + 1
50.        End If
51.    $t$ ← $t$ + 1
52.    $l$ ← $l$ + 1
53.    If $l$ = SN
54.        $l$ ← 1
55.    End If
56.    End If rand (0,1) < prob$_l$
57. End While
58. /*The scout phase*/
59. For $l$ = 1 to SN
60.       If (trial$_l$ = limit)
61.        Generate a new solution $X_l$ using (algorithm 2)
62.        Update TL using (17)
63.       End If
64.     End For
65.     Memorize the BGS
66. Until Cycle = maxCycle
67. Return BGS
```

The computational complexity of the MABC algorithm was determined by evaluating the worst-case time complexity of each component. The time complexity of the MABC algorithm was estimated to be $O((d_{emp}+d_{on})\times n_i\times m)$.

IV. COMPUTATIONAL RESULTS

This section assesses the performance of the MABC algorithm.

A. Problem instances

TABLE I
DETAILS OF E1(I01 – I20) AND E2 (INST01 – INST20)

Instance	n	n_i	m	N
I07	100	10	10	1000
I08	150	10	10	1500
I09	200	10	10	2000
I10	250	10	10	2500
I11	300	10	10	3000
I12	350	10	10	3500
I13	400	10	10	4000
INST01	50	10	10	500
INST02	50	10	10	500
INST03	60	10	10	600
INST04	70	10	10	700
INST05	75	10	10	750
INST06	75	10	10	750
INST07	80	10	10	800
INST08	80	10	10	800
INST09	80	10	10	800
INST10	90	10	10	900
INST11	90	10	10	900
INST12	100	10	10	1000
INST13	100	30	10	3000
INST14	150	30	10	4500
INST15	180	30	10	5400
INST16	200	30	10	6000
INST17	250	30	10	7500
INST18	280	20	10	5600
INST19	300	20	10	6000
INST20	350	20	10	7000

We experimentally examined the algorithms on two sets of benchmark instances: E1 (I07 – I13) proposed by Khan[52], and E2 (INST01 – INST20) proposed by Hifi *et al*.[57]. The instances (I01–I06) are known to be easily solvable in the literature. Hence, this study focused on the most difficult and large instances (E1 and E2).

All the benchmarks are available from the MMKP benchmarks website[82].

The first set contained 7 instances (I07–I13), and the number of groups in each instance varied between 100 and 400. Each group contained 10 items. Therefore, the number of decision variables ranges from 1000 to 4000. The second set contains 20 instances (INST01–INST20). The number of items in each group varied from 10 to 30 and the number of groups ranged from 50 to 400. Therefore, the total number of items in each instance of the second set varied from 500 to 7500.

All benchmarks are characterized by their common dimensionality size m = 10, n indicates the number of groups, and n_i indicates the size of group i. The details of these instances are summarized in Table I.

The best results are obtained using the following parameters

- maxCycle = 20
- limit = 5
- TL length = SN
- $d_{emp} = n / n_i$: Hamming distance (number of allowed exchanges) used in the employed-bee phase.
- $d_{on} = n_i$: hamming distance onlooker-bee phase

B. Performance of MABC

In this section, we compare the convergence performance of our proposed MABC algorithm, which includes surrogate relaxation, Hamming distance, and tabu list, with its basic version, BABC, which does not use these techniques. The objective of this comparative analysis is to demonstrate the favorable impact of these additional techniques on the convergence performance of the MABC algorithm.

Table II provides the detailed results for both algorithms for the two sets of instances. Column 2 shows the CPLEX solution. Columns 3 and 4 display the solutions obtained (Obj) by the BABC and its computational time (CPU), respectively, whereas columns 5 and 6 display the results for the MABC. Column 7 shows the deviation between the results of the two algorithms, given by

$$\%\text{dev} = \left(1 - \frac{\text{BABC}_{\text{MMKP}}}{\text{MABC}_{\text{MMKP}}}\right) \times 100 \qquad (18)$$

Columns 8 and 9 present the solutions obtained for the MABC when maxCycle was set to 2000. The best computational times and objective function values for the two algorithms are highlighted in bold.

The results in Table II indicate that BABC provides acceptable results within a short average computational time. The MABC attained better solutions than the BABC with shorter computational times. Therefore, MABC improves the instances by an average of 4.6% with a shorter execution time (a total of 3.46 s against 10.11 s). Evidently, the improvements in the results are due to modifications of the BABC to enhance its performance. Table II shows the good quality of the solutions provided by both BABC and MABC (7% on average for BABC and 2.4% on average for MABC over the quality of the solutions provided by CPLEX); the computational times for both algorithms are very low (0.1 s and 0.3 s on average for each instance for BABC and MABC, respectively), which proves that both the algorithms are very fast and adequate for real-time and critical-time problems. On average, the quality of the solutions generated by the MABC algorithm is 4.6% higher than that generated by the BABC algorithm. This improvement is ascribed to the modifications made during the

TABLE II
COMPARISON BETWEEN BABC AND MABC

Instance	CPLEX	BABC		MABC		% DEV	$MABC_{2000}$	
		Obj	CPU	Obj	CPU		OBJ	CPU
I07	24,595	23106	0.152	**24006**	**0.07**	4	24050	5.08
I08	36,884	34724	0.106	**36034**	**0.1**	4	36189	8.65
I09	49,176	46538	**0.147**	**48009**	0.2	3	48200	15.53
I10	61,475	57965	**0.131**	**60075**	0.3	4	60311	29.55
I11	73,783	69872	**0.13**	**72115**	0.4	3	72456	36.37
I12	86,091	81109	**0.14**	**84933**	0.5	5	84583	55.02
I13	98,437	92473	**0.164**	**96203**	0.6	4	96675	63.98
INST01	10738	9682	0.019	**10391**	**0.01**	7	10266	1.69
INST02	13598	12440	**0.02**	**13273**	**0.02**	6	13289	1.83
INST03	10946	10252	0.406	**10509**	**0.024**	2	10588	3.91
INST04	14449	12751	0.315	**13980**	**0.04**	9	13984	5.27
INST05	17059	16077	**0.026**	**16478**	0.04	2	16511	3.42
INST06	16835	15858	**0.026**	**16269**	0.03	3	16300	4.08
INST07	16444	14872	0.058	**15901**	**0.04**	6	15741	4.99
INST08	17507	16335	**0.029**	**16946**	0.04	4	16993	3.86
INST09	17759	16701	**0.028**	**17135**	0.04	3	17215	3.22
INST10	19307	17259	**0.035**	**18645**	0.05	7	18694	3.94
INST11	19441	17336	**0.037**	**18729**	0.05	7	18732	4.20
INST12	21731	19573	**0.034**	**20997**	0.06	7	21023	4.92
INST13	21577	20140	**0.107**	**20942**	0.2	4	20946	22.23
INST14	32871	30412	**0.132**	**31875**	0.5	5	31878	46.13
INST15	39160	36373	**0.162**	**37985**	0.7	4	38031	54.38
INST16	43364	40303	**0.181**	**42198**	0.9	4	42234	82.36
INST17	54360	50198	**0.23**	**52746**	1.3	5	52747	105.39
INST18	60465	56356	**0.187**	**58727**	0.9	4	58787	64.82
INST19	64929	60366	**0.223**	**63246**	1.3	5	63256	53.47
INST20	75616	70363	**0.238**	**73538**	1.7	4	73608	68.43
Average			0.128		0.37	4.6		
Sum			3.46		10.11			756.72

different phases of the basic version (utility ratio, surrogate relaxation, random exchanges, Hamming distance, TL, etc.).For the same limit parameter value, a higher maxCycle parameter value (maxCycle = 50) was set for the BABC. However, the computational time for the BABC algorithm was found to be (slightly) lower than that for the MABC algorithm. This can be ascribed to the fact that, in the BABC, only one item is exchanged randomly in the employed-bee and onlooker-bee phases. Therefore, the number of loops and mathematical computations were lower. Finally, we can conclude that the modified version achieved an improvement of almost 4.6% over the basic version, without any additional computational time.

Furthermore, different configuration parameters may provide better solution quality. However, this often leads to a significantly longer CPU time. Column 8 shows a deviation of 0.21% in the quality of the solutions obtained when the maxCycle was changed from 20 to 20×100 = 2000. The CPU time increased from less than 10 s for all instances to 756 s. The set of values chosen in our experiment showed an acceptable trade-off between the quality of the objective function and the required computational time.

C. Further analysis of MABC behavior

In this study, we conducted a sensitivity analysis of two crucial parameters, maxCycle and limit, to analyze the behavior of the MABC algorithm. First, we set the limit parameter to a predetermined value and varied the maxCycle parameter. Then, we fixed the maxCycle parameter and varied the limit parameter to evaluate the impact of each parameter on the performance of the algorithm. Through this approach, we gained insight into the optimal values of these parameters to achieve an efficient and effective optimization.

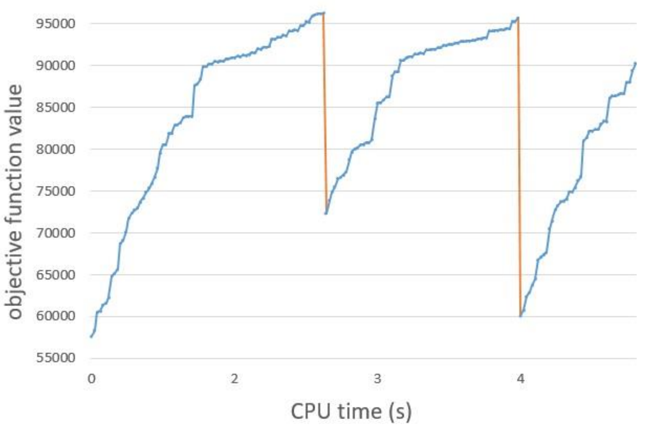

FIGURE 4. **Solution rate evolution of MABC.**

Fig. 4 provides a closer view of the behavior of the MABC by illustrating the evolution of the quality of solutions over time (I13 as an example). The figure demonstrates a continuous gradual improvement in the quality of the solutions over the cycles until it reaches a maximum (stagnation phase). When the threshold for the number of non-improvement (limit parameter) times is reached, the solution is discontinued and replaced by a new randomly chosen solution. This explains the sudden drop in the quality of solutions (orange lines).

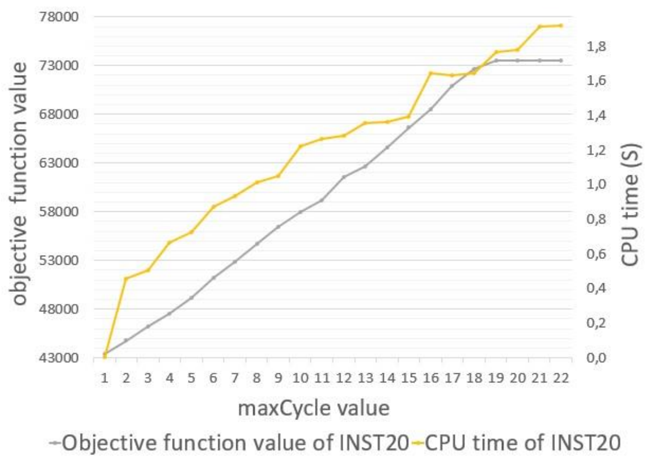

FIGURE 5. Behavior of MABC for instance 20 when varying maxCycle with the limit set at 5.

Fig. 5 shows the behavior of MABC when fixing the limit parameter value and varying the maxCycle parameter value for instance 20. It is clear from the figure that the objective function value of the obtained solutions gradually increases by 2.7% from maxCycle 1 to 19 (41% improvement in total), until it becomes almost constant at 20. This significant improvement requires almost no extra-computation time (less than 0.1 ms in each cycle).

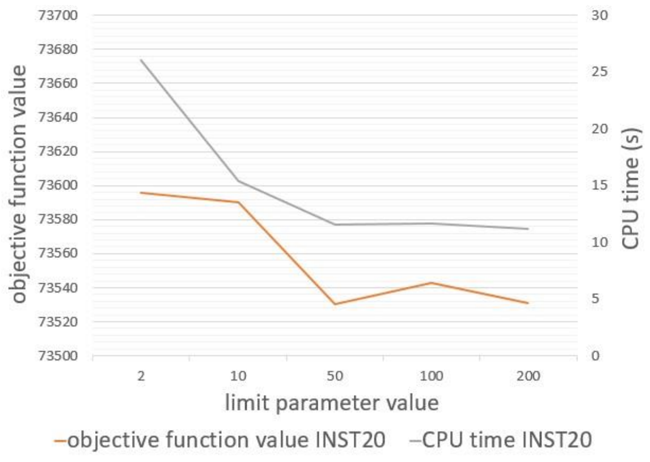

FIGURE 6. Fig. 6. Behavior of MABC for instance 20 when varying the limit and maxCycle set at 100.

Fig. 6 shows the sensitivity of the limit parameter to the quality of the obtained solution by setting the maxCycle parameter value and varying only the limit parameter for INST20. The sensitivity of the limit parameter variation becomes detectable only with a large number of cycles (maxCycle = 100 in this case) and a significant separation interval between the values of the limit parameter (intervals of 40, 50, and 100 in this case). Fig. 6 shows that when the limit value is relatively low (the solution quickly reaches the re-initialization threshold), the execution time becomes considerable (26 s for limit = 2 and maxCycle = 100), whereas by increasing the value of the limit, the execution time decreases gradually (until waiting for 11 s in this case). The decrease in CPU time is accompanied by a slight decrease in the quality of the objective function because the more the value of the limit parameter increases, the less the reinitialization phase (scout-bee phase) is executed, and consequently, the running time will be reduced (and vice versa).

D. Comparative study

In this subsection, we compare the results of the MABC with those of state-of-the-art algorithms.

We implemented MABC in Java JDK version 8, and all the reported computational experiments were conducted on a PC with a 2.30-GHz Intel i5 CPU.

Table III presents the results of our approach, compared to the results of six approaches from the literature, namely RLS: reactive local search-based algorithm[57], ALMMKP: ACO approach[63], PEGF-PERC: two variants of the reduce and solve approach[54], TIKS-TIKS^2: two variants of the two-phase iterative kernel search approach[74] and D-QPSO: Diversity reserved quantum particle swarm optimization approach[68]. The details of the running configurations of all these state-of-the-art algorithms, except those of ALMMKP (not reported), are presented next.

RLS: The algorithms were coded in C++ and tested on an Ultra-Sparc10 250 Mhz.

PEGF-PERC: The algorithms were executed on an Intel Xeon 2.83 GHz E5440 CPU and CPLEX 12.4. Two variants, PEGF and PERC, were proposed and the best results obtained are presented in Table III.

TIKS and TIKS^2: The algorithms were coded in Java 8 and Gurobi 9.0 is used as an MILP solver. The tests were executed on an Intel Core I7-5930K 3.5 GHz processor. TIKS has a 1200 s time limit using six cores and TIKS^2 has a 3600 s time limit using two cores on the machine.

D-QPSO: The algorithm was coded in MATLAB, run on an Intel Core 2 2.66GHz and tested only on the E1 benchmark set.

To enable comparisons between the best results, Table III presents the best objective function value and the minimal running time produced by our algorithm within 100 trials using different random seeds for each instance.

TABLE III
COMPARISON OF MABC WITH STATE-OF-THE-ART ALGORITHMS

Instance	CPLEX	MRLS HIFI		ALMMKP		PEGF-PERC		TIKS (6 CORES)		TIKS^2 (2 CORES)		D-QPSO		MABC		
		Obj	CPU	Obj	CPU	OBJ	CPU	OBJ	CPU	OBJ	CPU	OBJ	CPU	OBJ	CPU	% MAX
I07	24.595	24587	37	24557	3.125	24.590	75	**24595**	1200	24595	3600	24162	≈38	24006	**0.07**	2
I08	36.884	36877	37	36869	5.579	36.892	74	**36895**	1200	36894	3600	36403	≈50	36034	**0.1**	2
I09	49.176	49167	25	49156	6.815	49.181	393	49188	1200	**49189**	3600	48331	≈50	48009	**0.2**	2
I10	61.475	61437	47	61457	9.07	61.473	144	**61481**	1200	61480	3600	60432	≈100	60075	**0.3**	2
I11	73.783	73773	41	73775	10.506	73.787	205	**73792**	1200	73791	3600	72951	≈143	72115	**0.4**	2
I12	86.091	86069	42	86063	12.711	86.090	206	**86095**	1200	**86095**	3600	84890	≈170	84933	**0.5**	1
I13	98.437	98429	160	98420	14.933	98.436	212	**98443**	1200	**98443**	3600	97201	≈220	96203	**0.6**	2
INST01	10738	10714	10	10702	1.829	**10.738**	97	**10738**	1200	**10738**	3600	-	-	10391	**0.01**	3
INST02	13598	13598	76	13591	1.275	**13.598**	38	**13598**	1200	**13598**	3600	-	-	13273	**0.02**	2
INST03	10946	10943	58	10922	3.265	10.947	7	**10955**	1200	**10955**	3600	-	-	10509	**0.024**	3
INST04	14449	14429	8	14441	3.772	14.447	121	**14457**	1200	14456	3600	-	-	13980	**0.04**	3
INST05	17059	17053	42	17032	3.603	17.055	127	17061	1200	**17065**	3600	-	-	16478	**0.04**	3
INST06	16835	16823	50	16807	3.454	16.832	76	**16845**	1200	16838	3600	-	-	16269	**0.03**	3
INST07	16444	16423	65	16406	4.343	16.440	241	16442	1200	**16444**	3600	-	-	15901	**0.04**	3
INST08	17507	17506	27	17485	3.516	17.503	20	**17518**	1200	**17518**	3600	-	-	16946	**0.04**	3
INST09	17759	17754	51	17724	3.251	17.760	124	**17762**	1200	**17762**	3600	-	-	17135	**0.04**	3
INST10	19307	19314	32	19281	4.983	19.314	275	**19320**	1200	19318	3600	-	-	18645	**0.05**	3
INST11	19441	19431	111	19422	4.28	19.437	114	**19449**	1200	**19449**	3600	-	-	18729	**0.05**	3
INST12	21731	21730	23	21706	3.829	21.738	4	**21744**	1200	21743	3600	-	-	20997	**0.06**	3
INST13	21577	21569	18	21573	6.862	21.575	15	**21580**	1200	**21580**	3600	-	-	20942	**0.2**	2
INST14	32871	32869	72	32873	12.037	32.873	209	32874	1200	**32875**	3600	-	-	31875	**0.5**	3
INST15	39160	39148	63	39155	19.317	39.161	146	**39165**	1200	**39165**	3600	-	-	37985	**0.7**	3
INST16	43364	43354	194	43361	23.532	**43.367**	98	43366	1200	43366	3600	-	-	42198	**0.9**	2
INST17	54360	54349	30	54356	41.406	54.360	50	**54362**	1200	**54362**	3600	-	-	52746	**1.3**	2
INST18	60465	60456	201	60458	22.782	60.466	22	**60469**	1200	60468	3600	-	-	58727	**0.9**	2
INST19	64929	64921	45	64925	26.866	64.932	150	64931	1200	**64933**	3600	-	-	63246	**1.3**	2
INST20	75616	75603	47	75611	40.102	75.614	51	**75617**	1200	75616	3600	-	-	73538	**1.7**	2
Average			37		11.53		122		1200		3600				**0.37**	2.4
Sum			915		293.92		3219		32 400		97 200				**10.11**	

Note that different trials yielded slightly different results, with deviations in the range of 0.1–1.5% in the quality of the obtained solutions. Additionally, the best solutions obtained with CPLEX 12.9 within a time limit of 3600 s are reported.

Most heuristic comparisons in the literature are based on the objective function values. However, it is not fair to assess algorithms by comparing only the best reported solutions. Nevertheless, to facilitate an objective comparison between the computation times of the methods cited earlier, they must be run on the same platform and configuration. Because these algorithms are not available, this is not possible.

Therefore, our comparative study included both computational time and objective function values to better evaluate the performance of our algorithm. The best computational times and objective function values for the state-of-art algorithms, cited in table III, are highlighted in bold. The last column indicates the percentage of solutions achieved by our algorithm compared to the best objective function value from the literature. Finally, the last two rows of Table III, labeled as “Average” and “Sum,” report the runtime average and runtime sum, respectively, of all the solutions over all the instances realized by each of the considered methods.

The quality of the solutions generated by the MABC is evaluated against the CPLEX solutions in Table III, showing an average similarity of nearly 2%. This indicates that the MABC produces high-quality solutions that are close to the optimal solutions obtained by CPLEX. Our algorithm demonstrated a significantly improved runtime compared to other algorithms. In comparison to the fastest algorithm (ALMMKP) cited in Table III, which recorded an average runtime of 11.5 s per solution and a total runtime of 294 s for all instances, our algorithm exhibited an average runtime of 0.37 s per solution and a total runtime of 10.11 s for all solutions. Consequently, our algorithm, MABC, resulted in a time saving of more than 283 s for all instances when compared to the fastest algorithm from the literature (ALMMKP). Moreover, for the largest instance, INST17 (7500 variables), MABC provides a value close to 2%, similar to the best solution in less than 2 s (1.3 s).

To further illustrate the significance of the results obtained from the proposed MABC algorithm compared with those of other state-of-the-art algorithms (RLS, ALMMKP, PEGF-PERC, and TIKS), we performed a Kruskal-Wallis test, which is a nonparametric statistical test that compares the median values of independent groups based on the ranks of the observations. The test output provides the p-value and H statistic, where the former measures the probability of observing the data if the null hypothesis (no differences between the groups being compared) is true, and the latter measures the overall difference among the medians of the groups being compared.

We applied the Kruskal-Wallis test on two levels: the objective function value and CPU. For the objective function value, the H statistic was 0.31506, with a p-value of 0.98882, indicating no significant difference among the medians of the groups being compared. This suggests that there is no significant difference between the objective function values of the solutions obtained by the MABC algorithm and those obtained by the other state-of-the-art algorithms.

For CPU, the H statistic was 84.90850 and the p-value was 0, indicating strong evidence to reject the null hypothesis of equal medians among the compared groups. Thus, we can conclude that there is a significant difference in the CPU performance of the MABC algorithm compared with other state-of-the-art algorithms.

V. Conclusion

In this study, we developed a new approach (MABC) to solve the MMKP problem based on the ABC algorithm combined with surrogate relaxation, Hamming distance, and tabu list. The proposed method was validated using 27 benchmark instances. The experimental results verified that MABC generated competitive results (2.4% proximity to CPLEX solutions) within very short computational time (in milliseconds).

The Kruskal-Wallis test was used to compare the performance of the MABC algorithm with that of other state-of-the-art algorithms in terms of objective function value and CPU efficiency. Statistical analysis revealed no statistically significant difference between the objective function values obtained by the MABC algorithm and the other algorithms (H = 0.31506, p = 0.98882). However, for CPU performance, the Kruskal-Wallis test demonstrated a statistically significant difference between the MABC algorithm and the other algorithms (H = 84.90850, p = 0). Thus, it can be concluded that the MABC algorithm has a significantly different CPU efficiency compared with other state-of-the-art algorithms, with very short execution times.

However, increasing the parameter configuration values (limit and maxCycle parameters) may improve the quality of the obtained solutions, but with a significant computational time cost. Therefore, using the Hamming distance during the exploitation process to limit the local search to a finite number of groups (items) significantly reduces the time complexity of the algorithm. However, this may lead to a loss of information.

Therefore, the MABC algorithm is effective for solving optimization problems with complex constraints and objective functions. Nevertheless, the performance of the algorithm may depend on the selection of parameters, and it may not be suitable for problems requiring high precision or a large number of variables.

In future work, we aim to improve the quality of the obtained solutions without increasing the computational weight of the algorithm. This study was restricted to feasible exchanges between items. Thus, complex moves that traverse the solutions from the feasible search space to the infeasible space, and vice versa, can be examined. In addition, a systematic empirical study on the general swarm intelligence performance (including that of ABC and ant colony) can be conducted to achieve an efficient and consistent solution quality for the MMKP.

REFERENCES

[1] S. Khan, K. F. Li, E. G. Manning, and M. M. Akbar, "Solving the Knapsack Problem for Adaptive Multimedia Systems," *Studia Informatica Universalis*, vol. 2, pp. 157–178, 2002, [Online]. Available: http://studia.complexica.net/index.php?option=com_content&view=article&id=61:article-8&catid=35:number-1&Itemid=64&lang=en

[2] H. Pirkul, "An integer programming model for the allocation of databases in a distributed computer system," *Eur J Oper Res*, vol. 26, pp. 401–411, 1986.

[3] J. Tang, R. Yu, S. Liu, and J. L. Gaudiot, "A Container Based Edge Offloading Framework for Autonomous Driving," *IEEE Access*, vol. 8, pp. 33713–33726, 2020, doi: 10.1109/ACCESS.2020.2973457.

[4] C. Basnet and J. Wilson, "Heuristics for determining the number of warehouses for storing non-compatible products," *International Transactions in Operational Research*, vol. 12, no. 5, pp. 527–538, 2005, doi: 10.1111/j.1475-3995.2005.00523.x.

[5] H. Cao, X. Feng, Y. Sun, Z. Zhang, and Q. Wu, "A Service Selection Model with Multiple QoS Constraints on the MMKP," Apr. 2008, pp. 584–589. doi: 10.1109/npc.2007.35.

[6] C. Lee, J. Lehoezky, R. Rajkumar, and D. Siewiorek, "On quality of service optimization with discrete QoS options," *Real-Time Technology and Applications - Proceedings*, pp. 276–286, 1999, doi: 10.1109/RTTAS.1999.777680.

[7] M. Moghaddam and J. G. Davis, "Service selection in web service composition: A comparative review of existing approaches," *Web Services Foundations*, vol. 9781461475, pp. 321–346, 2013, doi: 10.1007/978-1-4614-7518-7_13.

[8] Q. Jiang, Y. Zhang, and J. Yan, "Neural Combinatorial Optimization for Energy-Efficient Offloading in Mobile Edge Computing," *IEEE Access*, vol. 8, pp. 35077–35089, 2020, doi: 10.1109/ACCESS.2020.2974484.

[9] S. Liang, Y. Li, Q. Dong, and X. Chen, "MMKP: A mind mapping knowledgebase prototyping tool for precision medicine," *Front Immunol*, vol. 13, Aug. 2022, doi: 10.3389/fimmu.2022.923528.

[10] D. Pisinger, "Budgeting with bounded multiple-choice constraints," *Eur J Oper Res*, vol. 129, no. 3, pp. 471–480, 2001, doi: 10.1016/S0377-2217(99)00451-8.

[11] J. White, B. Doughtery, and D. C. Schmidt, "ASCENT: An algorithmic technique for designing hardware and software in tandem," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 838–851, 2010, doi: 10.1109/TSE.2010.77.

[12] T. Wu, S. Zhang, X. Wu, and W. Dou, "A consumer-oriented service selection method for service-based applications in the cloud," *Proceedings - 16th IEEE International Conference on Computational Science and Engineering, CSE 2013*, pp. 838–845, 2013, doi: 10.1109/CSE.2013.127.

[13] F. Glover and G. A. Kochenberger, "CRITICAL EVENT TABU SEARCH FOR," *Meta-heuristics Springer, Boston, MA*, pp. 407–427, 1996.

[14] W. C. Healy, "Multiple Choice Programming (A Procedure for Linear Programming with Zero-One Variables)," *Oper Res*, vol. 12, no. 1, pp. 122–138, 1964, doi: 10.1287/opre.12.1.122.

[15] H. Kellerer, U. Pferschy, and D. Pisinger, "The multiple choice knapsack problem," pp. 317–347, 2004, doi: doi:10.1007/978-3-540-24777-7_11.

[16] D. Pisinger, "A minimal algorithm for the multiple-choice knapsack problem," *Eur J Oper Res*, vol. 83, no. 2, pp. 394–410, 1995, doi: 10.1016/0377-2217(95)00015-I.

[17] J. Puchinger, G. R. Raidl, and U. Pferschy, "The Multidimensional Knapsack Problem: Structure and Algorithms," *INFORMS J Comput*, vol. 22, no. 2, pp. 250–265, 2010, [Online]. Available: http://people.brunel.ac.uk/

[18] Q. H. K.S. Tang, K.F. Man, S. Kwong, "Genetic Algorithms and their applications," *IEEE Signal Process Mag*, no. November, pp. 22–37, 1996.

[19] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *IEEE International Conference on Neural Networks IV*, vol. 23, no. 2, pp. 1942–1948, 1995.

[20] M. Dorigo, M. Birattari, and T. Stuzle, "Ant colony optimisation," *IEEE Comput Intell Mag*, vol. 1, no. 4, pp. 28–39, 2006, doi: 10.1007/978-3-319-93025-1_3.

[21] Z. H. Zhan *et al.*, "An efficient ant colony system based on receding horizon control for the aircraft arrival sequencing and scheduling problem," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 2, pp. 399–412, Jun. 2010, doi: 10.1109/TITS.2010.2044793.

[22] D. Simon, "Biogeography-based optimization," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp. 702–713, 2008, doi: 10.1109/TEVC.2008.919004.

[23] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: Harmony Search," *Simulation*, vol. 76, no. 2, pp. 60–68, 2001, doi: 10.1201/b18469-3.

[24] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," *Technical report-tr06*, no. 2009, pp. 1–9, 2005.

[25] M. C. Yuen, S. C. Ng, M. F. Leung, and H. Che, "A metaheuristic-based framework for index tracking with practical constraints," *Complex and Intelligent Systems*, vol. 8, no. 6, pp. 4571–4586, Dec. 2022, doi: 10.1007/s40747-021-00605-5.

[26] R. Hongge, H. Jian, C. Wenbin, and X. Caihua, "Modeling and identification of rate-dependent and asymmetric hysteresis of soft bending pneumatic actuator based on evolutionary firefly algorithm, Mechanism and Machine Theory," *Mech Mach Theory*, vol. 181, no. 105169, 2023.

[27] S. Bitam, M. Batouche, and E. G. Talbi, "A survey on bee colony algorithms," in *Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum, IPDPSW 2010*, 2010, no. June 2014. doi: 10.1109/IPDPSW.2010.5470701.

[28] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *Journal of Global Optimization*, vol. 39, no. 3, pp. 459–471, 2007, doi: 10.1007/s10898-007-9149-x.

[29] D. Karaboga and B. Akay, "A comparative study of Artificial Bee Colony algorithm," *Appl Math Comput*, vol. 214, no. 1, pp. 108–132, 2009, doi: 10.1016/j.amc.2009.03.090.

[30] M. S. Kiran, H. Hakli, M. Gunduz, and H. Uguz, "Artificial bee colony algorithm with variable search strategy for continuous optimization," *Inf Sci (N Y)*, vol. 300, no. 1, pp. 140–157, 2015, doi: 10.1016/j.ins.2014.12.043.

[31] A. Banitalebi, M. I. A. Aziz, A. Bahar, and Z. A. Aziz, "Enhanced compact artificial bee colony," *Inf Sci (N Y)*, vol. 298, pp. 491–511, Mar. 2015, doi: 10.1016/j.ins.2014.12.015.

[32] D. C. Secui, "A new modified artificial bee colony algorithm for the economic dispatch problem," *Energy Convers Manag*, vol. 89, pp. 43–62, 2015, doi: 10.1016/j.enconman.2014.09.034.

[33] E. Kaya, B. Gorkemli, B. Akay, and D. Karaboga, "A review on the studies employing artificial bee colony algorithm to solve combinatorial optimization problems," *Eng Appl Artif Intell*, vol. 115, no. 105311, 2022.

[34] S. S. Choong, L. P. Wong, and C. P. Lim, "An artificial bee colony algorithm with a Modified Choice Function for the traveling salesman problem," *Swarm Evol Comput*, vol. 44, pp. 622–635, Feb. 2019, doi: 10.1016/j.swevo.2018.08.004.

[35] M. Alzaqebah, S. Abdullah, and S. Jawarneh, "Modified artificial bee colony for the vehicle routing problems with time windows," *Springerplus*, vol. 5, no. 1, Dec. 2016, doi: 10.1186/s40064-016-2940-8.

[36] A. S. Bhagade and P. v. Puranik, "Artificial Bee Colony (ABC) Algorithm for Vehicle Routing Optimization Problem," *International Journal of Soft Computing and Engineering*, vol. 2, no. 2, pp. 329–333, 2012.

[37] A. Draa and A. Bouaziz, "An artificial bee colony algorithm for image contrast enhancement," *Swarm Evol Comput*, vol. 16, pp. 69–84, 2014, doi: 10.1016/j.swevo.2014.01.003.

[38] T. Cura, "An artificial bee colony algorithm approach for the team orienteering problem with time windows," *Comput Ind Eng*, vol. 74, no. 1, pp. 270–290, 2014, doi: 10.1016/j.cie.2014.06.004.

[39] V. Coleto-Alcudia and M. A. Vega-Rodríguez, "Artificial Bee Colony algorithm based on Dominance (ABCD) for a hybrid gene selection method," *Knowl Based Syst*, vol. 205, Oct. 2020, doi: 10.1016/j.knosys.2020.106323.

[40] N. Arunachalam and A. Amuthan, "Integrated probability multi-search and solution acceptance rule-based artificial bee colony optimization scheme for web service composition," *Nat Comput*, vol. 20, no. 1, pp. 23–38, Mar. 2021, doi: 10.1007/s11047-019-09753-7.

[41] T. T. Aung, T. Thi, and S. Nyunt, "Discrete Artificial Bee Colony Algorithm for Community Detection in Social Networks," MERAL Portal, 2019.

[42] M. Alzaqebah and S. Abdullah, "Artificial bee colony search algorithm for examination timetabling problems," *International Journal of the Physical Sciences*, vol. 6, no. 17, pp. 4264–4272, 2011, doi: 10.5897/IJPS11.200.

[43] M. Alzaqebah and S. Abdullah, "An adaptive artificial bee colony and late-acceptance hill-climbing algorithm for examination timetabling," *Journal of Scheduling*, vol. 17, no. 3, pp. 249–262, 2014, doi: 10.1007/s10951-013-0352-y.

[44] M. Alzaqebah and S. Abdullah, "Hybrid Artificial Bee Colony Search Algorithm Based on Disruptive Selection for Examination Timetabling Problems," in *In Combinatorial Optimization and Applications: 5th International Conference, COCOA 2011*, Aug. 2011, vol. 6831, pp. 31–45.

[45] M. J. Mahmoodabadi and M. M. Shahangian, "A New Multi-objective Artificial Bee Colony Algorithm for Optimal Adaptive Robust Controller Design," *IETE J Res*, vol. 68, no. 2, pp. 1251–1264, 2022, doi: 10.1080/03772063.2019.1644211.

[46] Ş. Öztürk, R. Ahmad, and N. Akhtar, "Variants of Artificial Bee Colony algorithm and its applications in medical image processing," *Applied Soft Computing Journal*, vol. 97. Elsevier Ltd, Dec. 01, 2020. doi: 10.1016/j.asoc.2020.106799.

[47] W. Gao, S. Liu, and L. Huang, "A novel artificial bee colony algorithm based on modified search strategy and generalized opposition-based learning," *IEEE Trans Cybern*, vol. 43, no. 3, pp. 1011–1024, 2013, doi: 10.3233/IFS-141386.

[48] W. Bai, I. Eke, and K. Y. Lee, "An improved artificial bee colony optimization algorithm based on orthogonal learning for optimal power flow problem," *Control Eng Pract*, vol. 61, no. February, pp. 163–172, 2017, doi: 10.1016/j.conengprac.2017.02.010.

[49] J. Ng and D. Abbott, "Solid state quantum computers: A nanoscopic solution to the Moore's Law problem," 2001. [Online]. Available: http://spiedl.org/terms

[50] T. Ghasemi and M. Razzazi, "Development of core to solve the multidimensional multiple-choice knapsack problem," *Comput Ind Eng*, vol. 60, no. 2, pp. 349–360, 2011, doi: 10.1016/j.cie.2010.12.001.

[51] A. Sbihi, "A best first search exact algorithm for the Multiple-choice Multidimensional Knapsack Problem," *J Comb Optim*, vol. 13, no. 4, pp. 337–351, 2007, doi: 10.1007/s10878-006-9035-3.

[52] Khan, "Quality Adaptation in a Multisession Multimedia System: Model, Algorithms and Architecture," *Dissertation*. University of Victoria, Canada, p. 274, 1998.

[53] M. R. Razzazi and T. Ghasemi, "An exact algorithm for the multiple-choice multidimensional knapsack based on the core," *Communications in Computer and Information Science*, vol. 6 CCIS, no. 3, pp. 275–282, 2008, doi: 10.1007/978-3-540-89985-3_34.

[54] Y. Chen and J. K. Hao, "A 'reduce and solve' approach for the multiple-choice multidimensional knapsack problem," *Eur J Oper Res*, vol. 239, no. 2, pp. 313–322, 2014, doi: 10.1016/j.ejor.2014.05.025.

[55] M. , Moser, D. Jokanovic, and N. Shiratori, "An algorithm for the multidimensional multiple-choice knapsack problem," *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. 80, no. 3, pp. 582–589, 1997.

[56] M. M. Akbar, E. G. Manning, G. C. Shoja, and S. Khan, "Heuristic solutions for the multiple-choice multi-dimension knapsack problem," in *International Conference on Computational Science*, 2001, vol. 2074, pp. 659–668. doi: 10.1007/3-540-45718-6_71.

[57] M. Hifi, M. Michrafy, and A. Sbihi, "A reactive local search-based algorithm for the multiple-choice multi-dimensional knapsack problem," *Comput Optim Appl*, vol. 33, no. 2–3, pp. 271–285, 2006, doi: 10.1007/s10589-005-3057-0.

[58] N. Cherfi and M. Hifi, "Hybrid algorithms for the Multiple-choice Multi-dimensional Knapsack Problem," *International Journal of Operational Research*, vol. 5, no. 1, pp. 89–109, 2009, doi: 10.1504/IJOR.2009.024531.

[59] S. Hanafi and C. Wilbaut, "Improved convergent heuristics for the 0-1 multidimensional knapsack problem," *Ann Oper Res*, vol. 183, no. 1, pp. 125–142, 2011, doi: 10.1007/s10479-009-0546-z.

[60] R. Mansi, C. Alves, J. M. Valério De Carvalho, and S. Hanafi, "A hybrid heuristic for the multiple choice multidimensional knapsack problem," *Engineering Optimization*, vol. 45, no. 8, pp. 983–1004, 2013, doi: 10.1080/0305215X.2012.717072.

[61] C. Gao, G. Lu, X. Yao, and J. Li, "An iterative pseudo-gap enumeration approach for the Multidimensional Multiple-choice Knapsack Problem," *Eur J Oper Res*, vol. 260, no. 1, pp. 1–11, 2017, doi: 10.1016/j.ejor.2016.11.042.

[62] N. Cherfi and M. Hifi, "A column generation method for the multiple-choice multi-dimensional knapsack problem," *Comput Optim Appl*, vol. 46, no. 1, pp. 51–73, 2010, doi: 10.1007/s10589-008-9184-7.

[63] Z. Ren and Z. Feng, "An Ant Colony Optimization Approach to the Multiple-Choice Multidimensional Knapsack Problem," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation : Portland, Oregon, USA, July 07-11, ,* 2010, pp. 275–283.

[64] I. Crévits, S. Hanafi, R. Mansi, and C. Wilbaut, "Iterative semi-continuous relaxation heuristics for the multiple-choice multidimensional knapsack problem," *Comput Oper Res*, vol. 39, no. 1, pp. 32–41, 2012, doi: 10.1016/j.cor.2010.12.016.

[65] S. Htiouech, S. Bouamama, and R. Attia, "Using surrogate information to solve the multidimensional multi-choice knapsack problem," in *2013 IEEE Congress on Evolutionary Computation*, 2013, pp. 2102–2107.

[66] S. Htiouech and A. Alzaidi, "Smart Agents for the Multidimensional Multi-choice Knapsack Problem," *Int J Comput Appl*, vol. 174, no. 6, pp. 5–9, 2017, doi: 10.5120/ijca2017915404.

[67] Y. Xia, C. Gao, and J. L. Li, "A stochastic local search heuristic for the multidimensional multiple-choice knapsack problem," *Communications in Computer and Information Science*, vol. 562, pp. 513–522, 2015, doi: 10.1007/978-3-662-49014-3_46.

[68] H. Dong, X. Yang, X. Teng, and Y. Sha, "A diversity reserved quantum particle swarm optimization algorithm for MMKP," in *2016 IEEE/ACIS 15th International Conference on Computer and Information Science, ICIS 2016 - Proceedings*, Aug. 2016. doi: 10.1109/ICIS.2016.7550941.

[69] M. Caserta and S. Voß, "The robust multiple-choice multidimensional knapsack problem," *Omega (United Kingdom)*, vol. 86, pp. 16–27, 2019, doi: 10.1016/j.omega.2018.06.014.

[70] A. Mkaouar, S. Htiouech, and H. Chabchoub, "Solving the Multiple choice Multidimensional Knapsack problem with ABC algorithm," *2020 IEEE Congress on Evolutionary Computation, CEC 2020 - Conference Proceedings*, pp. 1–6, 2020, doi: 10.1109/CEC48606.2020.9185872.

[71] R. Mansini and R. Zanotti, "A Core-Based Exact Algorithm for the Multidimensional Multiple Choice Knapsack Problem," *INFORMS J Comput*, no. April, 2020.

[72] A. Syarif, D. Anggraini, K. Muludi, Wamiliana, and M. Gen, "Comparing various genetic algorithm approaches for multiple-choice multi-dimensional knapsack problem (mm-KP)," *International Journal of Intelligent Engineering and Systems*, vol. 13, no. 5, pp. 455–462, Oct. 2020, doi: 10.22266/ijies2020.1031.40.

[73] J. Yang, Y. H. Kim, and Y. Yoon, "A Memetic Algorithm with a Novel Repair Heuristic for the Multiple-Choice Multidimensional Knapsack Problem," *Mathematics*, vol. 10, no. 4, Feb. 2022, doi: 10.3390/math10040602.

[74] L. Lamanna, R. Mansini, and R. Zanotti, "A two-phase kernel search variant for the multidimensional multiple-choice knapsack problem," *Eur J Oper Res*, vol. 297, no. 1, pp. 53–65, Feb. 2022, doi: 10.1016/j.ejor.2021.05.007.

[75] A. Dellinger, Y. Lu, M. S. Song, and F. J. Vasko, "Simple strategies that generate bounded solutions for the multiple-choice multi-dimensional knapsack problem: a guide for OR practitioners," *International Transactions in Operational Research*, 2022.

[76] A. F. Achwak, B. M. Walid, B. Widad, B. Sarra, and S. Sara, “Algorithme génétique sophistiqué et schémas de parallélisation sur une grille de calcul pour la résolution du problème du sac à dos (Unbounded Knapsack),” 2016.
[77] G. Zhu and S. Kwong, “Gbest-guided artificial bee colony algorithm for numerical function optimization,” *Appl Math Comput*, vol. 217, no. 7, pp. 3166–3173, 2010, doi: 10.1016/j.amc.2010.08.049.
[78] A. Bookstein, V. A.Kulyukin, and T. Raita, “Generalized Hamming Distance,” *Inf Retr Boston*, vol. 5, pp. 353–375, 2002, doi: 10.1023/A.
[79] F. Glover, “A multiphase-dual algorithm for the zero-one integer programming problem.” pp. 879–919, 1965.
[80] C. J. Lin, M. S. Chern, and M. Chih, “A binary particle swarm optimization based on the surrogate information with proportional acceleration coefficients for the 0-1 multidimensional knapsack problem,” *Journal of Industrial and Production Engineering*, 2015, doi: 10.1080/21681015.2015.1111263.
[81] H. T. Jadhav and R. Roy, “Gbest guided artificial bee colony algorithm for environmental/economic dispatch considering wind power,” *Expert Syst Appl*, vol. 40, no. 16, pp. 6385–6399, 2013, doi: 10.1016/j.eswa.2013.05.048.
[82] “MMKP benchmarks,” 2012. http://www.es.ele.tue.nl/pareto/mmkp/.

ARIJ MKAOUAR received the B.Sc. degree in IT applied to industrial management and the M.Sc. degree in new information technologies and dedicated systems from the University of Sfax, Tunisia, in 2009 and 2012, respectively. She is currently pursuing the Ph.D. degree in information systems engineering at National Engineering School of Sfax, University of Sfax, Tunisia. Her research interests are optimization problems, artificial immune system, artificial bee colony algorithm and the multiple-choice multidimensional knapsack problem.

SKANDER HTIOUECH received the M.Sc. degree from the University of Valenciennes, Academy of Lille, France, and the Ph.D. degree from the National Engineering School of Tunis (ENIT), Tunisia. He is an assistant professor at the National Engineering School of Bizerte (ENIB), University of Carthage, Tunisia. Since 2016 he joined the University of Jeddah, KSA. His research interests are in the areas of modeling and optimization problems, linear programming, surrogate, lagrangian information, smart agent, multiple-choice multidimensional knapsack problem and crowd management and analysis via drones.

HABIB CHABCHOUB received a Ph.D. degree in administration sciences from Laval University, Canada. He is the Founding Director of the High Institute of Industrial Management at the University of Sfax, Tunisia. He is currently a Full Professor at the School of Business at Al Ain University, United Arab Emirates. His research focuses on multi-criteria decision aid, logistics and transport, mathematical programming, and operations management. He has published many articles in different international journals. He chaired sessions and conferences at the national and international levels. He acts as a referee in several journals.