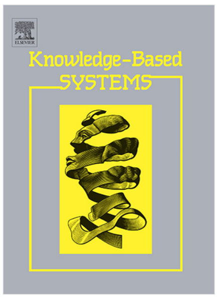
Evolutionary and adaptive inheritance enhanced Grey Wolf Optimization algorithm for binary domains

İlker Gölcük, Fehmi Burcin Ozsoydan



PII: S0950-7051(20)30066-6
DOI: https://doi.org/10.1016/j.knosys.2020.105586
Reference: KNOSYS 105586

To appear in: *Knowledge-Based Systems*

Received date : 21 August 2019
Revised date : 26 January 2020
Accepted date : 28 January 2020

Evolutionary and adaptive inheritance enhanced grey wolf optimization algorithm for binary domains

İlker Gölcük[1], Fehmi Burcin Ozsoydan[2]

Abstract

This paper introduces a new binary Grey Wolf Optimization (GWO) algorithm, which is one of the recent swarm intelligence-based metaheuristic algorithms. Its various extensions have been reported in the related literature. Despite numerous successful applications in real-valued optimization problems, the canonical GWO algorithm cannot directly handle binary optimization problems. To this end, transformation functions are commonly employed to map the real-valued solution vector to the binary values; however, this approach brings about the undesired problem of spatial disconnect. In this study, evolutionary and adaptive inheritance mechanisms are employed in the GWO algorithm so as to operate in the binary domain directly. To mimic the leadership hierarchy procedure of the GWO, multi-parent crossover with two different dominance strategies is developed while updating the binary coordinates of the wolf pack. Furthermore, adaptive mutation with exponentially decreasing step-size is adopted to avoid premature convergence and to establish a balance between intensification and diversification. The performance of the proposed algorithm is tested on the well-known binary benchmark suites comprised of the Set-Union Knapsack Problem (SUKP) that extends the 0-1 Knapsack Problem and the Uncapacitated Facility Location Problem (UFLP). Comprehensive experimental study including real-life applications and statistical analyses demonstrate the effectiveness of the proposed algorithm.

Keywords: Binary optimization; grey wolf optimization; knapsack problem; facility location problem; multi-parent crossover

Nomenclature

Abbreviation	
PSO	Particle Swarm Optimization
ACO	Ant Colony Optimization
FA	Firefly Algorithm
ABC	Artificial Bee Colony
WOA	Whale Optimization Algorithm
FPA	Flower Pollination Algorithm
CS	Cuckoo Search
GWO	Grey Wolf Optimization
KPCA	Kernel Principal Component Analysis
MP-Xover	Multi-Parent Crossover
SUKP	Set-Union Knapsack Problem
UFLP	Uncapacitated Facility Location Problem
WSA	Weighted Superposition Attraction
GA	Genetic Algorithms

[1] *Corresponding Author*: Department of Industrial Engineering, İzmir Bakırçay University, İzmir 35665, Turkey, e-mail: ilker.golcuk@bakircay.edu.tr, Phone: 00902324930000, Fax: 00902328447122
[2] Department of Industrial Engineering, Faculty of Engineering, Dokuz Eylül University, İzmir 35397, Turkey, e-mail: burcin.ozsoydan@deu.edu.tr, Phone: 00902323017630, Fax: 00902323017608

GWO_{fbd}	GWO algorithm with Fitness-based Dominance
GWO_{rbd}	GWO algorithm with Rank-based Dominance
S-GROA	Greedy Repairing and Optimization Algorithm
BABC	Binary Artificial Bee Colony Algorithm
binDE	Binary Differential Evolution
gPSO	Genetic-Particle Swarm Optimization
bWSA	Binary Weighted Superposition Attraction Algorithm
intAgents	Intelligent Agents
DJaya	Discrete Jaya
CPSO	Continuous Particle Swarm Optimization
prLeGWO	Priority and Learning Based Grey Wolf Optimization
DOE	Design of Experiments
S/N	Signal-to-Noise
RPD	Relative Percentage Deviation
ARDP	Absolute Relative Percentage Deviation
ANOVA	Analysis of Variance
CPU	Central Processing Unit
ATM	Automated Teller Machine
Knapsack Problem	KP

1. Introduction

Swarm intelligence is one of the emerging research fields of the past decade. It brings about new opportunities in solving real-world optimization problems. The vigorous development of swarm intelligence algorithms has led researchers to deal with non-linear, non-convex, and combinatorial optimization problems. Many state-of-the-art algorithms have been proposed by mimicking the swarm behavior of various species. Particle Swarm Optimization (PSO) [1], Ant Colony Optimization (ACO) [2], Firefly Algorithm (FA) [3], Artificial Bee Colony (ABC) [4], Whale Optimization Algorithm (WOA) [5], Flower Pollination Algorithm (FPA) [6], and Cuckoo Search (CS) [7] are among the well-known and promising swarm intelligence-based metaheuristic algorithms.

Grey Wolf Optimization (GWO) has recently been proposed by Mirjalili, Mirjalili and Lewis [8]. GWO is based on the leadership hierarchy of grey wolves and simulates the hunting behavior of wolf packs in nature. In recent years, substantial growth in the GWO related studies has been acknowledged in the literature. Luo, Zhang and Fan [9] studied energy-efficient flexible job shop scheduling problems with variable processing times. Oliveira, Oliveira, Boaventura-Cunha and Pinho [10] implemented chaotic GWO to optimize the parameters of a robust higher-order sliding modes controller. Singh and Singh [11] proposed a modified position update equations of the GWO, and real-valued function optimization problems were solved. Ozsoydan [12] investigated the effects of dominant wolves in GWO, and learning hierarchy curves with prioritization strategies were proposed. Jiang and Zhang [13] applied GWO for solving job shop and flexible job shop scheduling problems. Wang, Chen, Li, Cai, Zhao, Tong, Li and Xu [14] optimized the parameters of kernel extreme machines by using the GWO algorithm, and the proposed model was applied to bankruptcy prediction. Mirjalili, Saremi, Mirjalili and Coelho [15] extended the GWO algorithm to deal with multi-criterion optimization problems. Mirjalili [16] employed the GWO algorithm for training multi-layer perceptrons, and the performance of the GWO algorithm was compared with state-of-the-art solvers.

Although GWO has been successfully implemented in continuous-domain problems, most of the real-life problems, particularly in the field of business analytics, economics and logistics, require dealing with binary variables. Thus, one can conclude that binary optimization problems are omnipresent in real-life applications. Text document clustering [17-19], information retrieval systems [20], text feature selection [21, 22], 0-1 knapsack problem [23, 24], project portfolio selection [25], facility location problems [26] are some of the most frequently encountered application areas of binary optimization problems. Unfortunately, the canonical GWO algorithm cannot be directly used in such problems. In the related literature, there are some reported studies modifying GWO to be able to handle binary domains. In one of them, Too, Abdullah, Saad, Ali and Tee [27] used sigmoid transfer functions to convert real-valued variables into binary values in the GWO algorithm and applied binary GWO for feature selection in the EMG signal classification. Pamshetti, Singh and Singh [28] studied conservation voltage reduction and optimized network configuration via binary GWO algorithm. The sigmoid function is used as a transfer function to obtain binary values in the GWO algorithm. Veena, Arivazhagan and Jebarani [29] integrated GRASP with a binary GWO algorithm for improved detection of steganographic algorithms. The feature selection part was performed based on binary GWO, and the sigmoid function was used to transform a real-valued solution vector into binary digits. Velliangiri [30] hybridized kernel principal component analysis (KPCA) with a binary GWO algorithm for intrusion detection. The sigmoid transfer function was used to map real-valued numbers to binary solution vector. Al-Tashi, Kadir, Rais, Mirjalili and Alhussian [31] proposed a new hybrid algorithm combining PSO and GWO, and the proposed algorithm employed sigmoid transfer function to obtain binary values for feature selection problem.

Even though the sigmoid function has been widely adopted within binary GWO, transfer functions suffer from the problem of spatial disconnect [32]. The underlying assumption of a population-based algorithm is that close-by solutions in the continuous search space have similar fitness values so that when a real-valued solution vector is mapped into a binary vector, its fitness value is expected to be similar. However, solutions that are close to each other in the continuous domain may become far apart in the binary solution space. Thus, establishing a direct correspondence between continuous and binary spaces by using transfer functions is quite challenging. This disadvantage is prevalent in the related literature. Another observation is that direct manipulation of binary solution vectors results in better convergence trend towards high-quality solutions. It can be put forward that the proposed algorithm falls into the latter category.

In this study, the GWO algorithm has been extended to binary spaces by using evolutionary and adaptive inheritance mechanisms. The main advantage and novelty of the proposed algorithm are that there is no need to employ transfer functions because evolutionary operators work with the binary solution vectors to explore the search space. Since dominance hierarchy is one of the building blocks of the position updating mechanism in GWO, evolutionary operators in place of commonly used transfer functions are employed by the proposed approach due to the previously discussed shortcomings of transfer functions. To this end, a multi-parent crossover operation (MP-Xover) is designed to update binary coordinates of wolves. For the selection mechanism, fitness-based and rank-based dominance weights have been used to mimic the leadership hierarchy of the wolf pack. Moreover, an adaptive mutation has been carried out after MP-Xover operation in order to avoid premature convergence and to establish a balance between diversification and intensification. The performance of the proposed binary GWO algorithm has been tested on the well-known binary benchmark problems comprised of the Set-Union Knapsack Problem (SUKP) [33] and Uncapacitated Facility Location Problem (UFLP) [34]. The benchmark suites of the SUKP and UFLP are adopted from [35] and the OR-Lib (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/) respectively. The obtained results indicate that the proposed approach achieves promising performance in comparison with the state-of-the-art bio-inspired metaheuristic algorithms. Finally, proposed binary GWO algorithms are tested through a real-life ATM deployment problem [36] and an advertisement selection problem [37]. Statistical tests have also verified the effectiveness of the proposed algorithms.

The rest of the paper is organized as follows: The SUKP and UFLP problems are given in Section 2. Methodology and the proposed algorithm are introduced in Section 3. Experimental results are given in Section 4. Finally, concluding remarks are given in Section 5. For the convenience of the readers, all of the abbreviations used throughout the presented study are given with their full names in the nomenclature, as reported in [38].

2 Related Work

2.1 Set Union Knapsack Problem

Binary optimization problems have grabbed considerable attention from the research community in recent years. The knapsack problem is one of the well-known members of the binary optimization problems and has numerous real-life applications such as capital budgeting, resource allocation, cargo loading, etc. One can notice that the set Union Knapsack Problem (SUKP) is a generalization of the well-known 0-1 knapsack problem. Moreover, SUKP can be regarded as a generalization of the densest k-subhypergraph problem [39]. It can be formulated as in the following:

$$\begin{aligned} &\text{Max}\, P(A) = \sum_{i \in A} p_i \\ &s.t.\;\; W(A) = \sum_{j \in \bigcup_{i \in A} E_i} w_j \le C,\;\; A \subset I \end{aligned} \tag{1}$$

where $E = \{1, 2, \dots, n\}$ corresponds to set of n elements where each element $j\,(j = 1, 2, \dots, n)$ has a weight of $w_j > 0$, $I = \{1, 2, \dots, m\}$ denotes set of items where each item $i\,(i = 1, 2, \dots, m)$ corresponds to a subset of elements $I_i \subset I$, and C represents the knapsack capacity. The subsets of elements are given by a relation matrix, and each item is associated with a profit p_i. For an arbitrary non-empty item set $A \subset I$, the aim is to find a subset $A^* \subset I$ to maximize the total profit $\sum_{i \in I} p_i$ without exceeding the knapsack capacity given by $W(A^*) = \sum_{j \in \bigcup_{i \in A^*} E_i} w_j$.

Additionally, He, Xie, Wong and Wang [35] proposed an alternative mathematical formulation for the SUKP problem. In this formulation, the *m*-dimensional 0-1 solution vector is denoted by $Y = [y_1, y_2, \dots, y_m] \in \{0,1\}^m$. Given $A_Y = \{i \,|\, y_i \in Y, y_i = 1, 1 \le i \le m\} \subseteq I$, for an arbitrary $i \in I$, $y_i = 1$ if and only if $i \in A_Y$. In mathematical terms, the SUKP can be given as:

$$\begin{aligned} &\text{Max } f(Y) = \sum\nolimits_{i=1}^{m} y_i p_i \\ &s.t.\ W(A_y) = \sum_{j \in \bigcup_{i \in A_y} E_i} w_j \le C,\ \ A_y \subset I \end{aligned} \tag{2}$$

Goldschmidt, Nehme and Yu [33] generalized the 0-1 knapsack problem and introduced SUKP. Arulselvan [39] presented the necessary mathematical proofs, which contributed to the subsequent studies in terms of theoretical basis. Ozsoydan and Baykasoglu [40] integrated GA and PSO for binary optimization problems, and the SUKP benchmark suite was used to test the performance of the proposed method. Wu and He [41] proposed a hybrid Jaya algorithm with double coding for SUKP. Feng, An and Gao [42] evaluated the effect of transfer functions in solving SUKP. He, Xie, Wong and Wang [35] proposed a binary artificial bee colony algorithm for SUKP. Baykasoğlu, Ozsoydan and Senol [43] employed Weighted Superposition Attraction (WSA) algorithm for solving binary optimization problems. He and Wang [44] proposed a group theory-based optimization algorithm for solving SUKP.

2.2 Uncapacitated facility location problem

Facility location problems have been extensively studied in the optimization literature. UFLP is one of the well-known facility location problems, which have grabbed remarkable attention from the research community in recent years. In the UFLP, each customer is served by exactly one facility under the assumption of unlimited capacity [34]. The UFLP aims to determine the location of facilities by minimizing the total cost, which is the sum of facility opening costs and shipment costs. The mathematical formulation of the UFLP problem can be given as follows:

$$Minimize \sum_{k \in K} \sum_{l \in L} c_{kl} z_{kl} + \sum_{k \in K} b_k y_k \tag{3}$$

$$\sum_{k \in K} z_{kl} = 1,\ l \in L \tag{4}$$

$$z_{kl} \le y_k,\ k \in K, l \in L \tag{5}$$

$$z_{kl} \in \{0,1\},\ k \in K \tag{6}$$

$$y_k \in \{0,1\},\ k \in K \tag{7}$$

where K and L represents a set of facility locations and demand points, respectively. The opening cost of a facility at *k*-th location is denoted by b_k , the shipment cost between the facility opened at *k*-th location, and the *l*-th demand point is given by c_{kl}. The binary variable z_{kl} takes the value of 1 if *l*-th customer is served by the facility opened at the *k*-th location; otherwise, it takes the value of 0. Another binary variable is y_k that equals to 1 if the facility is opened at *k*-th location; otherwise, it takes the value of 0. In Eq. 3, the total cost is minimized. Eq. 4 states that each customer is served by exactly one facility. Eq. 5 ensures that only the opened facilities can serve to the customers. Finally, Eqs. 6-7 impose restrictions on the decision variables.

Metaheuristic and heuristic methods have been widely employed to solve UFLP. Jaramillo, Bhadury and Batta [45] used genetic algorithms to solve location problems. Al-Sultan and Al-Fawzan [46] employed the tabu search algorithm to solve UFLP. Wang, Wu, Ip, Wang and Yan

[47] used a multi-population based PSO algorithm to solve UFLP. Some PSO applications were developed by Güner and Şevkli [48] and Şevkli and Güner [49]. Tsuya, Takaya and Yamamura [50] implemented the firefly algorithm for UFLP. Kıran [51] employed the artificial bee colony algorithm to solve UFLP. In another work, Baykasoğlu, Ozsoydan and Senol [43] adopted WSA to solve this problem. Recently, Ozsoydan [52] proposed artificial search agents with cognitive intelligence for a set of binary optimization problems, including UFLP. More detailed literature surveys regarding facility location problems and solution approaches can be found in [53, 54].

2.3 The standard GWO

GWO algorithm was developed by Mirjalili, Mirjalili and Lewis [8] to solve real-valued, constrained, and unconstrained optimization problems. GWO algorithm draws inspiration from the social hierarchy of grey wolves that artificial wolves mimic the behaviors of tracking, encircling, attacking, etc. According to the social hierarchy of GWO, there are four types of wolves, which are referred to as alpha (α), beta (β), delta (δ), and omega (ω) wolves. The best solution is considered as α wolf. The second- and the third-ranked solutions are regarded as β and δ wolves, respectively. The hunting behavior, which represents optimization procedures, is guided by α, β, δ wolves. The rest of the population is considered as ω and they are influenced by the movements of those three dominant wolves. The dominance hierarchy of a wolf pack is visualized in Fig. 1. According to the dominance hierarchy, the α wolf commands all the subordinates. Similarly, β and δ wolves impact the wolves under their social positions.

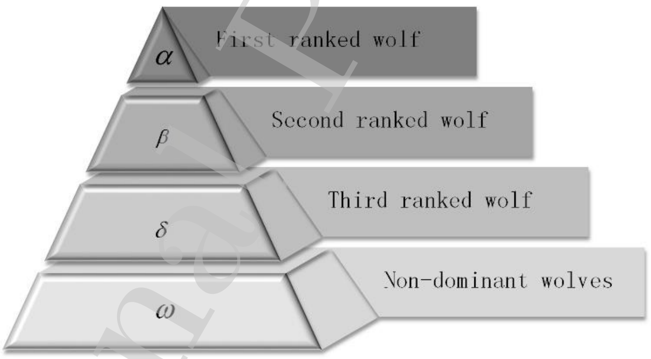


Fig. 1. Hierarchal order of grey wolves in nature

Hunting behavior of grey wolves begins with encircling of prey. In mathematical terms, the encircling behavior is modeled by using the following equations:

$$\vec{D} = \left| \vec{C} \,.\, \vec{X}_p(t) - \vec{X}(t) \right| \tag{8}$$

$$\vec{X}(t+1) = \vec{X}_p(t) - \vec{A} \,.\vec{D} \tag{9}$$

where t represents iterations, $\vec{A}$ and $\vec{C}$ are coefficients vectors, $\vec{X}_p$ is the position vector of the prey and X is the position vector of a grey wolf. The coefficient vectors are calculated as:

$$\vec{A} = 2\vec{a} \,.\vec{r}_1 - \vec{a} \tag{10}$$

$$\vec{C} = 2\ .\vec{r}_2 \tag{11}$$

The component of the coefficient vector $\vec{a}$ is linearly decreased from 2 to 0 as $a(t) = 2 - (t-1)\cdot\left(2/maxIter\right)$ that the maximum number of iterations is given by $maxIter$.

The encircling of prey is followed by hunting behavior. The movement of the grey wolves is controlled by dominant wolves (α, β, δ) . In this regard, the hunting behavior of the wolf pack is formulated by Eqs. 12-18 in the simulated environment. Finally, a pseudo-code for the canonical GWO algorithm is presented by Algorithm 1.

$$\vec{D}_\alpha = \left|\vec{C}_1\ .\ \vec{X}_\alpha - \vec{X}\right| \tag{12}$$

$$\vec{D}_\beta = \left|\vec{C}_2\ .\ \vec{X}_\beta - \vec{X}\right| \tag{13}$$

$$\vec{D}_\delta = \left|\vec{C}_3\ .\ \vec{X}_\delta - \vec{X}\right| \tag{14}$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1\ .\vec{D}_\alpha \tag{15}$$

$$\vec{X}_2 = \vec{X}_\beta - \vec{A}_2\ .\vec{D}_\beta \tag{16}$$

$$\vec{X}_3 = \vec{X}_\delta - \vec{A}_3\ .\vec{D}_\delta \tag{17}$$

$$\vec{X}\left(t+1\right) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \tag{18}$$

Algorithm 1. A pseudo-code for the canonical GWO.

```
1:  initialize GWO population of solution vectors
2.  define coefficient vectors $\vec{A}$ , $\vec{C}$ and $\vec{a}$
3:  evaluate fitness values $f\left(x_i\right)$ , $i = 1,\dots,nWolf$
4:  determine $\vec{X}_\alpha$ , $\vec{X}_\beta$ and $\vec{X}_\delta$
5:  while iter < maxIter
6:          for i = 1:nWolf
7:            movement of grey wolves, according to Eq. 11
8:          end for
9:          Update coefficients $\vec{A}$ , $\vec{C}$ and $\vec{a}$
10:         calculate fitness values $f\left(x_i\right)$ , $i = 1,\dots,n$
11:         update $\vec{X}_\alpha$ , $\vec{X}_\beta$ and $\vec{X}_\delta$
12:         iter ← iter +1
13: end while
14: return $\vec{X}_\alpha$, $f\left(\vec{X}_\alpha\right)$
```

2.4 Genetic algorithm

Genetic algorithms (GA) are inspired by the fundamental mechanisms of the evolution and Darwinian theory. Each possible solution to the problem is represented as a chromosome. Then evolutionary search operators are used to guide the optimization process. GA is considered as a member of evolutionary and population-based stochastic search algorithms.

The standard GA begins with randomly generated solutions based on the predefined problem boundaries. Then the fitness of each solution is calculated. Afterward, the selection operation is performed either at random or by using a particular procedure such as roulette-wheel, tournament selection, etc. The selected chromosomes undergo crossover and mutation operations so as to generate offspring population. Next, fitness values of the offspring are obtained, and finally, natural selection is invoked again to select the individuals of the next generation either form the enlarged sampling pool that is comprised of both parents and offspring or only from the offspring population. Optionally, the elitism scheme can be implemented in order to keep the best-found solutions in the population. A pseudo-code for the standard GA algorithm is given in Algorithm 2.

Algorithm 2. A pseudo-code for GA

```
 1:set the parameters of GA
 2:generate initial population
 3:evaluate fitness values of individuals
 4:while (stopping criterion is not met) do
 5:      select individuals for reproduction
 6:      perform crossover
 7:      perform mutation
 8:      evaluate fitness of offspring
 9:      select individuals for the next generation
10:      keep the elite individual (optional)
11:end while
12:return the best solution and associated fitness value
```

3. Proposed binary GWO algorithms

The proposed binary GWO algorithm is comprised of three main steps, which are generating the initial population, the evolution of the wolf pack, and handling infeasible solutions, in sequence. In the subsequent sections, these steps are explained in detail. The general procedures of the proposed binary GWO algorithm are depicted in Fig. 2. The next section is devoted to reporting the basics of generating an initial population.

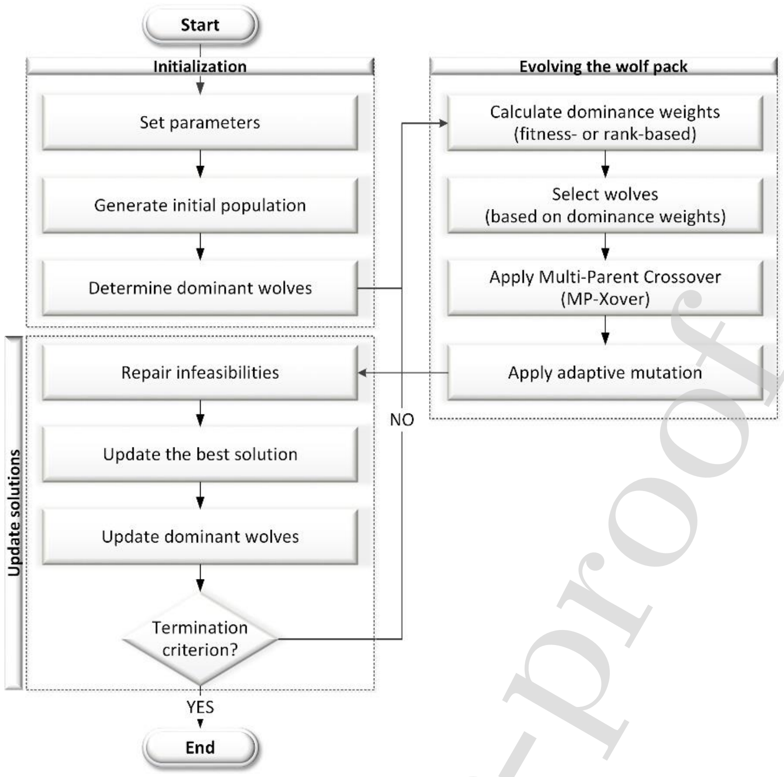


Fig. 2. General procedures of the proposed binary GWO

3.1 Generating initial population

The proposed algorithm starts with generating an initial population randomly. Since each bit can take only binary values, equal probabilities are assigned to each value at random to generate the initial solution. More formally, the population of wolves $\vec{X}_i = \{x_{i,1}, x_{i,2}, \cdots, x_{i,j}, \cdots x_{i,nDim}\}$ is generated based on Eq. 19.

$$x_{i,j} = \begin{cases} 1, & \text{if } rand < 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

where $x_{i,j}$ denotes the binary value at the *j*-th position of the *i*-th wolf.

After generating the first wolf pack, the next step is to search for the neighborhood solutions. Because the GWO algorithm has initially proposed for the real-valued function optimization problems, it cannot be directly applied to binary domains. Therefore, a search mechanism is needed to search in binary spaces, and at the same time, this mechanism is expected to be well-suited to the original metaphor and notions of the GWO algorithm. In this study, evolutionary and adaptive inheritance mechanisms are employed in order to carry out search in the binary landscape. Thereby, MP-Xover and adaptive mutation mechanisms are used to evolve the generated binary wolf pack. In the next section, these mechanisms are introduced in detail.

3.2 Evolving the wolf pack

The evolution of the wolf pack is a crucial step in the proposed algorithm. According to the canonical GWO algorithm, three dominant wolves, namely α, β, δ, determine the search behavior of the algorithm. Each wolf is influenced by the dominant wolves, and the position of each wolf is the joint product of the contribution of the wolf itself and the three dominant wolves. Therefore, the proposed search mechanism uses MP-Xover to search for neighboring binary solutions. Moreover, population diversity is maintained by using an adaptive mutation scheme, which is also to be clarified in the following sections.

3.2.1 Multi-parent crossover

The MP-Xover can be seen as a generalization of uniform crossover [55]. MP-Xover goes by several other names in the literature, such as multi-sexual crossover [56], scanning crossover [57], and gene pool recombination [58]. In default, the arity of the crossover operator in GA is two. From the technical point of view, the arity of the crossover operator can be more than two, and even can have an arity from one up to the population size (*nWolf*) [59]. In the MP-Xover, the number of parents is more than two. In this study, parents are determined as α, β, δ wolves and the wolf itself. Therefore, there are 4 parents in the present implementation. One should also note that one-child or multiple-children can be produced as a result of MP-Xover. As the position of the *i*-th-wolf is determined by using MP-Xover operator, one-child is generated after each MP-Xover operation. Let the produced child be represented by $y = \{y_1, y_2, \cdots y_j, \cdots, y_{nDim}\}$, the *j*-th dimension of the child can be given as:

$$y_j = x_{s,j} \tag{20}$$

where $j \in [1, nDim]$ and s is a subset represented by $s \in \{1, \cdots, 4\} \subseteq [1, nWolf]$.

The critical question related to implementation choice arises here: how to select parents from the recombination pool for each bit. In other words, whether each parent delivers its allele with different frequencies or not.

In this study, the chance of being chosen is determined by two different weighting approaches, depending on the dominance strategies, which are defined as fitness-based dominance and rank-based dominance. To this end, fitness values of α, β, δ wolves and the *i*-th wolf are calculated. In fitness-based dominance, the chance of being selected is proportional to fitness. Therefore, dominance weights are taken as the normalized fitness values. The pseudo-code related to the calculation of fitness-based dominance weights is given in Algorithm 3.

Algorithm 3. *Fitness-based dominance weights*

```
Input: $\vec{X}_\alpha, \vec{X}_\beta, \vec{X}_\delta, \vec{X}_i$
Output: $\vec{W} = \{w_1, w_2, w_3, w_4\}$
1: $f_{accum} \leftarrow 0$
2: for $k$ = 1:4 do
3:         $f_k \leftarrow fitness(\vec{X}_k)$  // $\{1:\alpha, 2:\beta, 3:\delta, 4:i\}$
4:         $f_{accum} \leftarrow f_{accum} + f_k$
5: end for
6: for $k$ = 1:4 do
7:         $w_k \leftarrow f_k / f_{accum}$
8: end for
9: return $\vec{W}$
```

According to Algorithm 3, each wolf compares its fitness value with the dominant wolves, and fitness values are transformed into dominance weights after normalizing with the accumulated fitness values (f_{accum}).

On the other hand, rank-based dominance utilizes ranking orders of the dominant wolves and the wolf itself, rather than fitness values. The pseudo-code related to the calculation of rank-based dominance weights is given in Algorithm 4.

Algorithm 4. *Rank-based dominance weights*

```
Input: $\vec{X}_\alpha, \vec{X}_\beta, \vec{X}_\delta, \vec{X}_i, \tau$
Output: $\vec{W} = \{w_1, w_2, w_3, w_4\}$
1:for $k = 1:4$ do
2:    $f_k \leftarrow fitness(\vec{X}_k)$ // $\{1:\alpha, 2:\beta, 3:\delta, 4:i\}$
3:end for
4: $ranks \leftarrow \arg sort_k \{f_k\}$, $k \in \{1,2,3,4\}$ // Sort from best to worst
5:for $k = 1:4$ do
6:    $w_k \leftarrow (rank_k)^{-\tau}$
7:end for
8: $w_k = w_k / \sum_k w_k$, $k \in \{1,2,3,4\}$
9:return $\vec{W}$
```

These ranking orders are transformed into rank-based dominance weights as given in Eq. 21.

$$w_k = (rank_k)^{-\tau},\ k \in \{1,2,3,4\} \tag{21}$$

where w_k denotes the weight of the *k*-th wolf, *k* being the α, β, δ and the *i*-th wolf, respectively. The variable *rank* indicates the ranking order of the *k*-th wolf. The parameter τ is used to control dominance weights, where smaller τ values tend to give an equal chance of being selected. The parameter τ also exhibits biased and unbiased behaviors in that τ value of 0 gives each wolf an equal chance to deliver its allele. The effect of τ parameter on the rank-based dominance weights is visualized in Fig. 3.

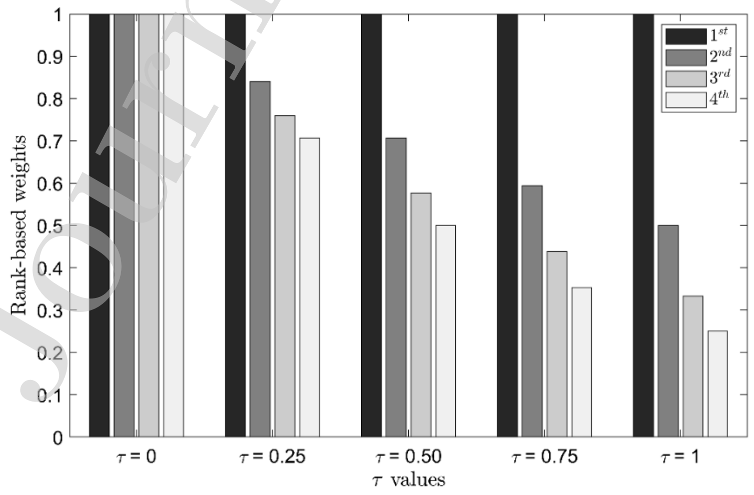


Fig. 3. Effect of τ values on the rank-based dominance weights

Once the dominance weights are determined, the selection operation is performed to deliver the alleles of the wolves to the child. The pseudocode of the selection operation is given in Algorithm 5. The input of the parent selection algorithm is the dominance weights found in the earlier step. Here, w_1, w_2, w_3, w_4 denote the dominance weights of α, β, δ and the *i*-th wolf, respectively.

Algorithm 5. *Selection for the MP-Xover*

```
Input: $\vec{W} = \{w_1, w_2, w_3, w_4\}$ // normalized weights
Output: $\vec{X}_{selected} = \{x_{selected,1}, x_{selected,2}, \cdots, x_{selected,nDim}\}$
1:Generate a uniform random number $r \in [0,1]$
2: $w_{accum} \leftarrow 0$
3: $k \leftarrow 0$
4:while $w_{accum} < r$ do
5:     $k \leftarrow k+1$
6:     $w_{accum} \leftarrow w_{accum} + w_k$
7:end while
8: $\vec{X}_{selected} \leftarrow \vec{X}_k$
9:return $\vec{X}_{selected}$
```

For each bit of the solution vector, the selection operator decides on the wolf that delivers its allele. The binary GWO algorithm utilizing fitness-based dominance weights is denoted as GWO_{fbd}. If the rank-based dominance weights are used, then the proposed binary GWO algorithm is named GWO_{rbd}. The MP-Xover operation is visualized in Fig. 4.

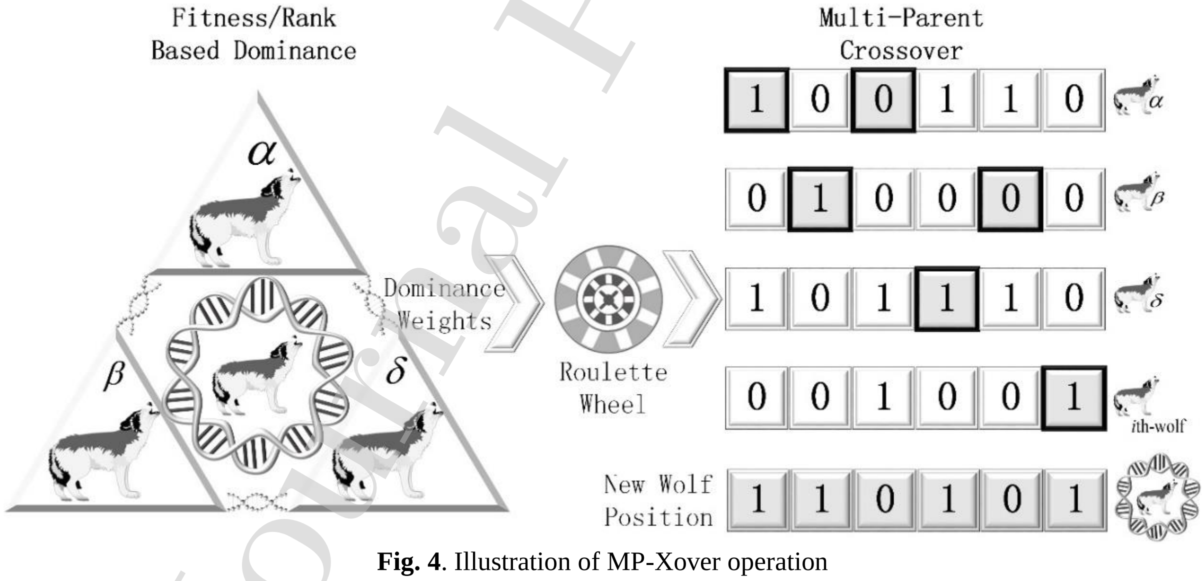


Fig. 4. Illustration of MP-Xover operation

As can be seen from Fig. 3, first and the third bits are taken from the α wolf, second and fifth bits are delivered from the β wolf, and fourth and sixth bits are transferred to a child from the δ and *i*-th wolves, respectively. MP-Xover operation is followed by the adaptive mutation operation, as discussed in the next section.

3.2.2 Adaptive mutation

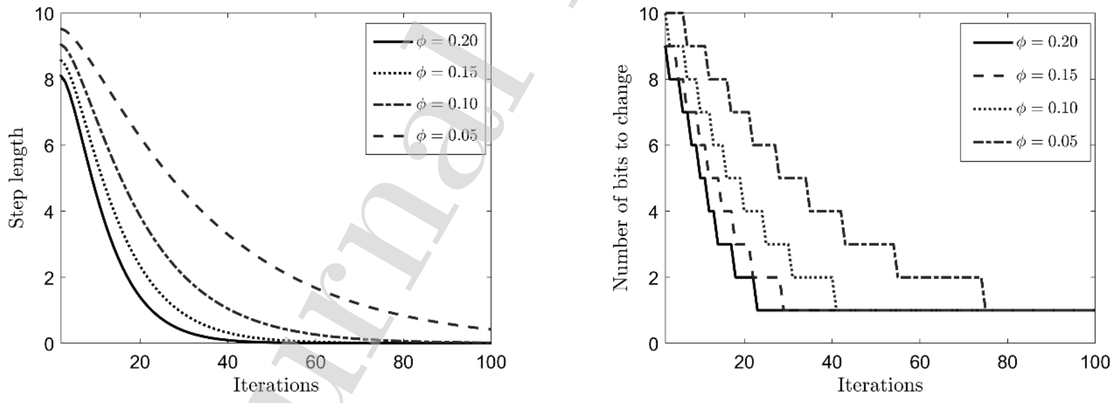
Adaptive mutation operation is an important mechanism to avoid premature convergence of the population. The original GWO algorithm does not provide any auxiliary mechanisms to circumvent the problem of reduced diversity within the population. In order to enhance the search capability of the proposed binary GWO algorithm, mutation operation has been adopted. In the mutation operation, randomly selected bits are flipped. The number of bits being flipped is the primary design asset in the adaptive mutation scheme.

The number of bits to change is determined based on an exponentially decreasing step-length [40, 60, 61]. In order to determine step-length, the parameter *initRate* $(r_{t=1})$ is defined, which denotes the initial value for the step-length. This value is gradually decreased throughout generations by using a user-defined parameter ϕ (Phi) that controls the speed of decrement. After exponentially decreasing the step-size, it is multiplied by the number of dimensions and rounded up to obtain the total number of bits to flip. The adaptive binary step-length is calculated as given in Eq. 22-23.

$$r_{t+1} = r_t - e^{-t/(t+1)} \times \phi \times r_t \tag{22}$$

$$bits2\ flip_{t+1} = \lceil r_{t+1} \times nDim \rceil \tag{23}$$

At the beginning of the search procedure, the adaptive mutation mechanism provides greater diversification as the variable *bits2flip* is at its maximum. Then, it is exponentially decreased during iterations, which yield more intensified search around the promising regions towards the end of the generations. The behavior of exponentially decreasing step-size is illustrated in Fig. 5.



a) Continuous step-length b) Binary step-length

Fig. 5. Exponentially decreasing step-length for adaptive mutation

The number of bits to change is affected by the initial value (*initRate*) of the step-length and the parameter ϕ. There is a delicate balance between these two parameters, and they lend themselves to parameter-tuning. In the experimental study, parameter-tuning will be covered in detail.

3.3 Handling infeasibility

Another issue is to deal with infeasible solutions. In this study, infeasible solutions are generated only in the SUKP as the solution can exceed the knapsack capacity. The UFLP does not fall into the infeasible region so that a particular procedure is adopted only for the SUKP.

Infeasible solutions are handled via S-GROA algorithm, originally proposed by He, Xie, Wong and Wang [35]. S-GROA algorithm uses profit and density information and calculates the value density of each item. Then, all items are sorted in descending order according to value densities, and the indices are stored. The violating items are eliminated, and the solution is reconstructed in a greedy-manner by using the obtained value densities. Moreover, if there is unused remaining capacity in the knapsack, S-GROA performs additional improvement and assigns items to the knapsack to improve utilization. Because S-GROA is widely used in the benchmark problems in the literature, this procedure is adopted in this study without any modification. Further details can be found in [35].

To sum up, the proposed GWO algorithm can be given in Algorithm 6.

Algorithm 6. *Proposed Binary GWO algorithm*

```
Input: nWolf, maxIter, $initRate, \phi, \tau$
Output: $\vec{X}_\alpha$ and associated fitness value
 1:set parameters of binary GWO
 2:generate initial binary wolf pack and evaluate fitness
 3:calculate adaptive step-size
 4:determine dominant wolves
 5:while $(iter < maxIter)$ do
 6:     for i = 1:nWolf do
 7:         Generate empty new solution $\vec{X}_{new} = \{x_{new,1}, x_{new,2}, \cdots, x_{new,nDim}\}$
 8:         /* Multi-parent Crossover */
 9:         // Get dominance weights based on fitness-based or rank-based dominance
10:         $\vec{W} = getDominanceWeights(\vec{X}_\alpha, \vec{X}_\beta, \vec{X}_\delta, \vec{X}_i)$ // run Algorithm3 or 4
11:         for j = 1:nDim do
12:             // Select dominant wolf for multi-parent crossover
13:             $\vec{X}_{selected} = selectWolfMP\text{-}XOver(\vec{W})$ // run Algorithm 5
14:             $x_{new,j} = x_{selected,j}$
15:         end for
16:         /*Mutation Operation */
17:         for j = 1:bits2flip(iter) do
18:             Select a random bit $randIdx \in [1, nDim]$
19:             $x_{new,randIdx} = 1 - x_{new,randIdx}$
20:         end for
21:         /* Handling infeasibility */
22:         $\vec{X}_{new} \leftarrow S-GROA(\vec{X}_{new})$ // specific to SUKP
23:         $\vec{X}_i \leftarrow \vec{X}_{new}$
24:         /* Update Best Solutions */
25:         evaluate fitness of $\vec{X}_i$
26:         update the best solution
27:         update dominant wolves
28:     end for
```

```
29:        $iter \leftarrow iter + 1$
30:end while
31:return $\vec{X}_{\alpha}$ and best fitness
```

4. Experimental Results

In the experimental study, benchmark suites of the SUKP and UFLP are used to test the performance of the proposed algorithms. The first set of instances belongs to SUKP, which has been adopted from [35]. The instances of SUKP are represented as $m\_n\_\alpha\_\beta$, where m represents the number of items, n is the number of elements, α is the density of ones in the relation matrix, and β is the ratio of knapsack capacity to the sum of weights of all elements. There are three groups of instances in the SUKP. In the first benchmark suite, there are 10 instances with $m > n$ which is also denoted as *Fi1*, *Fi2*, …, *Fi10*. In the second set of instances, $m = n$ and they are represented as Se1, Se2, …, Se10. Finally, in the third set of instances, $m < n$ and these instances are named Thi1, Thi2, …, Thi10. All of the benchmark suites are solved by using Matlab R2019a, and the results are reported.

The parameters are taken from the literature as they are reported to ensure fair comparisons with the previous SUKP studies. That is to say, the population size (*nWolf*) is set to 20, and the termination criterion is determined to be the maximum number of iterations (*maxIter*). The maximum number of iterations is taken as $\max\{m, n\}$ as in the [35, 40, 52]. All the results are evaluated over 100 independent replications. The proposed algorithms are compared with the state-of-the-art methods reported in the literature. For the SUKP problem, the compared algorithms can be given as follows: A-SUKP [35] , GA [35], BABC [35], ABCbin [35], binDE [35], bWSA [43], gPSO* [40], gPSO [40], intAgents [52], Djaya [41]. Additional parameters of the algorithms are given as follows: In the GA, single-point crossover, uniform mutation, and roulette wheel selection are employed with crossover and mutation probabilities being 0.8 and 0.01, respectively. In BABC and ABCbin, *a = 5.0, limit = Max(m,n)/5*. In binDE, scaling factor *F* is taken as 0.5 and the crossover constant *CR=0.3*. In the bWSA, $\tau = 0.8$, $\phi = 0.008$, and *CR=0.8*. In gPSO* and gPSO algorithms, $r^1 = 0.05$, $\phi = 0.005$, $p_1 = 0.10$, $p_2 = 0.70$. In the intAgents, crossover parameters are set to 0.60 and 0.10, and mutation parameters are taken as 0.005 and 0.5, respectively. Finally, Djaya algorithm takes the crossover rate as 0.8, scaling factor as 0.8, and Cauchy mutation parameter as being 1.

The second benchmark suite is the UFLP instances, which are retrieved from the OR-lib[3]. Because optimum solutions are known in these benchmark problems, how many times the best solutions are found during the course of replications is used as a performance metric. This performance metric is called "hit values" and considered as the main benchmarking criterion in the related literature. As reported in the literature, *nWolf* is set to the number of facilities in the instance, and *maxIter* is determined as 1000. For the UFLP problem, the compared algorithms are as follows: CPSO [62], ABC_{bin} [63], bWSA [43], PSO [12], GWO [12], prLeGWO [12], intAgents [52]. The population size is taken as the number of facilities, and the maximum number of iterations is set to 1000 for all of the algorithms. CPSO sets cognitive coefficients as $c_1=c_2=0.2$, and inertia weight is given as a random number between 0.5 and 1. In the ABC_{bin},

[3] http://people.brunel.ac.uk/~mastjjb/jeb/info.html

the *limit* parameter is calculated as $m \times n$. In the bWSA, $\tau = 0.8$, $\phi = 0.008$, and *CR=0.8*. PSO algorithm sets *c1=c2=2* and v_{min}*=-6*, v_{max} *= 6*. In the GWO algorithm, the coefficient *a* is gradually decreased from 2 to 0. The prLeGWO algorithm sets initial values of w_{alpha}, w_{beta}, w_{delta} as 1/3, and ultimate values as 0.80, 0.10, 0.10, respectively. Finally, in the intAgents, crossover parameters are set to 0.60 and 0.10, and mutation parameters are taken as 0.005 and 0.5, respectively.

It is important to emphasize that the same number of function evaluations are used to compare the algorithms. Making a fair comparison between population-based algorithms is a critical subject in the field of metaheuristic optimization. The algorithms should be compared under the same conditions. Fortunately, the literature of the SUKP and UFLP provides a well-defined procedure to ensure making fair comparisons among different population-based algorithms. The number of populations and the maximum number of iterations are precisely defined in the literature, in which all of the cited papers adopt these parameters in the same way. Table 1 summarizes the number of consumed function evaluations for the SUKP and UFLP.

Table 1. Total number of function evaluations in the SUKP and UFLP instances

problem	*population size*	*maximum number of iterations*	*total number of function evaluations*
SUKP	20	max(*m*,*n*) in the dataset*	20 x max*(m,n)*
UFLP	Number of facilities in the dataset	1000	1000 x number of facilities

* *m*: number of items, *n*: number of elements in the dataset.

In the SUKP problem, the population size is fixed to 20. Moreover, the maximum number of iterations is defined as the maximum of two parameters given in the dataset, which are the number of items and number of elements. As the items and elements are represented by *m* and *n*, respectively, the total number of function evaluations in the SUKP is given by $20 \times \max\{m, n\}$.

On the other hand, the related literature of the UFLP problem dictates that the population size should be set to the number of facilities, which is given in the instance of the dataset. Furthermore, the number of iterations is taken as 1000 in all of the compared algorithms. Consequently, the total number of function evaluations in the UFLP is given by $1000 \times \text{number of facilities}$.

Because the above-mentioned parameter settings are being used exactly the same way in the literature, we have confidently compared the performance of the proposed algorithms with their counterparts.

4.1 Parameter calibration of the proposed binary GWO

In this section, a detailed design of experiment (DOE) study is carried out in order to calibrate the parameters of the proposed binary GWO. The full factorial design is one of the most widely used DOE approaches that examine all possible combinations of parameter values. However, it becomes an arduous task to perform all the experiments when the number of factors is relatively large. The Taguchi method, on the other hand, requires much fewer experiments to be executed in comparison with the full factorial design. Taguchi method makes use of orthogonal arrays and divides the factors into two categories: controllable factors and noise factors. The noise factors are those whose outcomes are not controllable. Because the elimination of the noise factors is often impractical, the Taguchi method tries to minimize the effects of noise. To this end, optimal levels of significant controllable factors are determined, relying on the concept of robustness. Signal-to-Noise (S/N) ratio is measured in which the terms signal and noise designate the mean of the response variable and standard deviation, respectively. In the Taguchi method, three different objective functions are addressed, which are smaller-the-better, nominal-is-the-best, and larger-the-better.

In the Taguchi analysis, the design factors are determined as initial mutation rate (*InitRate)*, the parameter which controls the slope of the decreasing function of mutation rate *(Phi)*, and parameter of the rank-based weights *(Tau)*. The population size is set to 20 a prior in order to make fair comparisons with the reported results in the literature. Because GWO_{fbd} utilizes only *InitRate* and *Phi,* which are standard parameters in both GWO_{fbd} and GWO_{rbd}, the parameters are calibrated based on GWO_{rbd} algorithm. Preliminary study has been conducted, and five levels for each factor are determined. The parameter levels are determined as $InitRate \in \{0.1, 0.3, 0.5, 0.8, 1\}$, $Phi \in \{5E-5, 5E-4, 5E-3, 5E-2, 5E-1\}$, $Tau \in \{0, 0.25, 0.50, 0.75, 1\}$. Note that the tau parameter of zero corresponds to equal selection probabilities in the MP-Xover operation. With the three factors and five levels for each factor, L25 is selected as the fittest orthogonal design which fulfills our minimum requirements. The factor levels of the orthogonal array are given in Table 2.

Table 2. Factor levels of orthogonal array L25

Treatment No	Factor Levels		
	InitRate	Phi	Tau
1	0.10	5.E-05	0.00
2	0.10	5.E-04	0.25
3	0.10	5.E-03	0.50
4	0.10	5.E-02	0.75
5	0.10	5.E-01	1.00
6	0.30	5.E-05	0.25
7	0.30	5.E-04	0.50
8	0.30	5.E-03	0.75
9	0.30	5.E-02	1.00
10	0.30	5.E-01	0.00
11	0.50	5.E-05	0.50
12	0.50	5.E-04	0.75
13	0.50	5.E-03	1.00
14	0.50	5.E-02	0.00
15	0.50	5.E-01	0.25
16	0.80	5.E-05	0.75
17	0.80	5.E-04	1.00
18	0.80	5.E-03	0.00
19	0.80	5.E-02	0.25
20	0.80	5.E-01	0.50
21	1.00	5.E-05	1.00
22	1.00	5.00E-04	0.00
23	1.00	5.00E-03	0.25
24	1.00	5.00E-02	0.50
25	1.00	5.00E-01	0.75

Because SUKP is a challenging combinatorial optimization problem, parameter calibration is performed on selected small-, medium-, and large-scaled SUKP instances. To this end, 100_85_0.15_0.85 (Fi2), 200_185_0.15_0.85 (Fi4), 285_300_0.15_0.85 (Th6), 300_300_0.15_0.85 (Se6), 500_500_0.15_0.85 (Se10) instances are employed in the parameter calibration. As the test problems have different sizes and objective functions are incommensurate, objective function values are transformed into Relative Percentage Deviation (RPD). RPD values are calculated for each instance-treatment pair as follows:

$$RDP_i = \frac{Max_{sol} - GWO_{sol_i}}{Max_{sol}} \times 100 \qquad (24)$$

where GWO_{sol_i} is the objective function value obtained by the GWO algorithm in treatment *i*, and Max_{sol} is the maximum objective function value found among 25 treatments. (Note that SUKP is a maximization problem that the profit is being maximized).

When the RPD values are calculated for each instance-treatment pair, average RDP values (ARDP) are obtained for each treatment by averaging RDP values over the considered problem instances. Then, the S/N ratios are computed based on the ARDP values. The objective function is selected as the smaller-the-better type. Fig. 6 shows the main effects plot for the parameters of binary GWO.

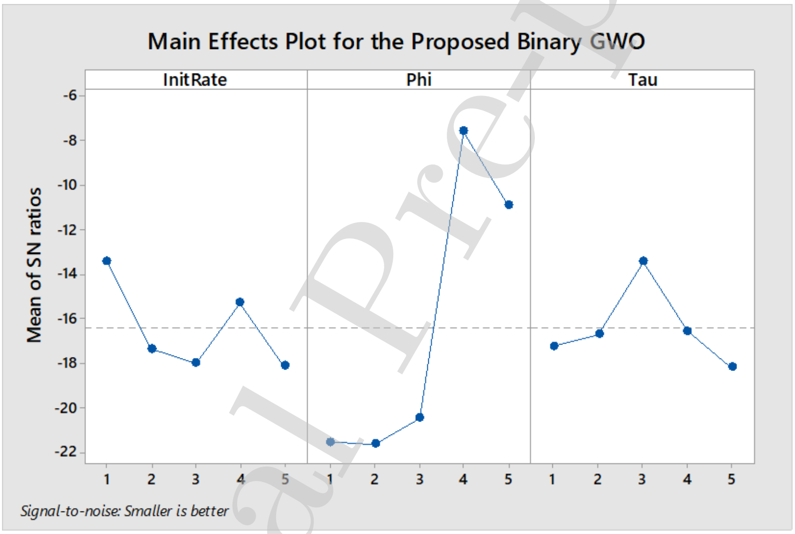


Fig. 6. Main effects plot for the parameters of GWO_{rbd}

According to the main effects plot shown in Fig. 6., *InitRate* level of 0.1, *Phi* level of 5E-2, and *Tau* level of 0.50 yield the highest S/N ratios. The main effects plot does not give statistical evidence to demonstrate a significant difference between levels parameters. Therefore, ANalysis Of VAriance (ANOVA) is conducted for the S/N ratios and ARDP values. If the significant difference cannot be obtained between S/N ratios, the ARDP values are considered as the second criteria [64, 65]. Before applying ANOVA, three main hypotheses, namely normality, homogeneity of variance, and independence, are tested and validated. The resulting ANOVA table is given in Table 3.

Table 3. ANOVA results for the parameters of binary GWO

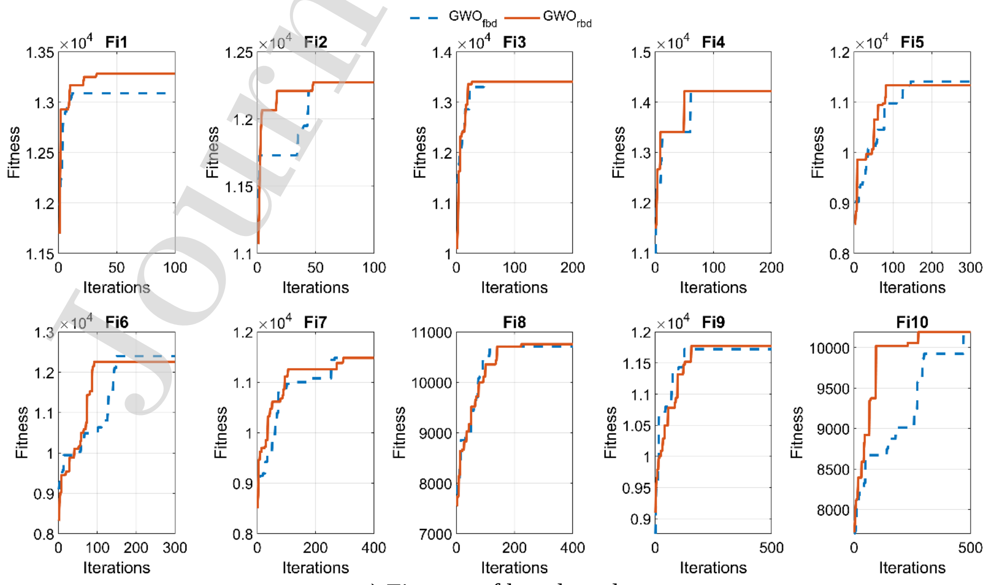
	parameter	*DF*	*Adj SS*	*Adj MS*	*F-value*	*p-value*	*significance level*	
							0.05	0.10
ANOVA results for S/N ratios	*InitRate*	4	83,48	20,870	2,55	0,093	~	+
	Phi	4	897,77	224,442	27,46	0,000	+	+

	Tau	4	63,85	15,964	1,95	0,166	~	~
	Error	12	98,07	8,173				
	Total	24	1143,17					
ANOVA results for ARDP values	*InitRate*	4	30,45	7,611	3,36	0,046	+	+
	Phi	4	460,24	115,059	50,78	0,000	+	+
	Tau	4	65,42	16,355	7,22	0,003	+	+
	Error	12	27,19	2,266				
	Total	24	583,29					

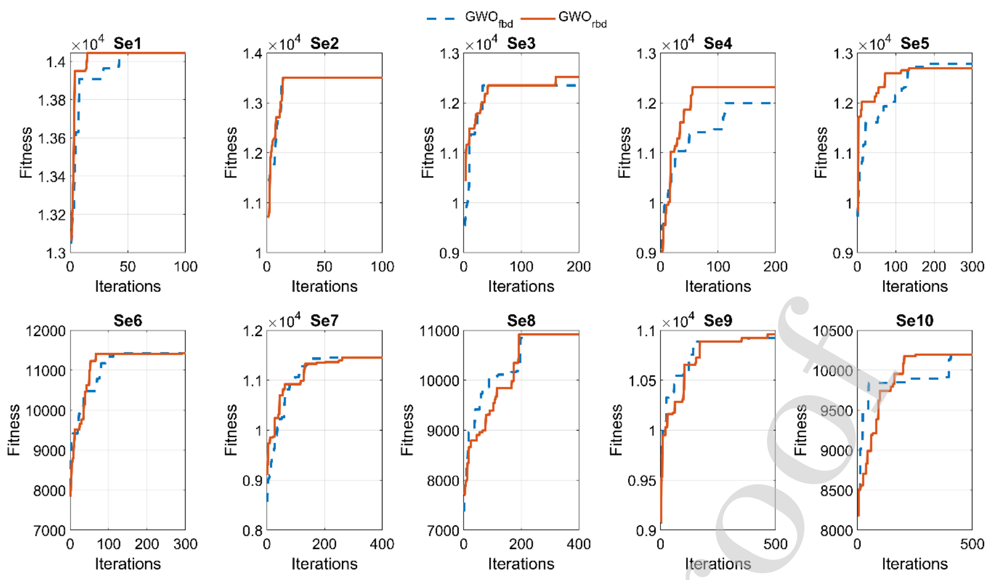
In Table 2, *DF* represents the degree of freedom, the *Adj SS* indicates the adjusted sum of squares, and the *adj MS* represents adjusted mean squares. The greater *F*-value means that the considered factor has more effect on the response variable. The *p*-value shows the statistical significance in that if the *p*-value is less than or equal to the desired significance level, one can assure that there is a statistical difference between the levels of the considered factor. In this study, the significance level is determined as 0.05 and 0.10. According to ANOVA results for the S/N ratios, levels of *Phi* parameter are statistically different for the significance level of 0.05. On the other hand, levels of *InitRate* and *Phi* are statistically different for the significance level of 0.10. On the other hand, ANOVA results for the ARDP values show that there are statistical differences between levels of *InitRate*, *Phi*, and *Tau* parameters for the significance levels of 0.05 and 0.10. Note that the largest F-values are obtained for the parameter *Phi* so that setting the right parameter value for the Phi has a significant impact on the response variable. Finally, based on the Taguchi analysis, the following parameter values are determined: *InitRate* = 0.1, *Phi* = 5E-2, and *Tau* = 0.50.

4.2 Computational results for SUKP

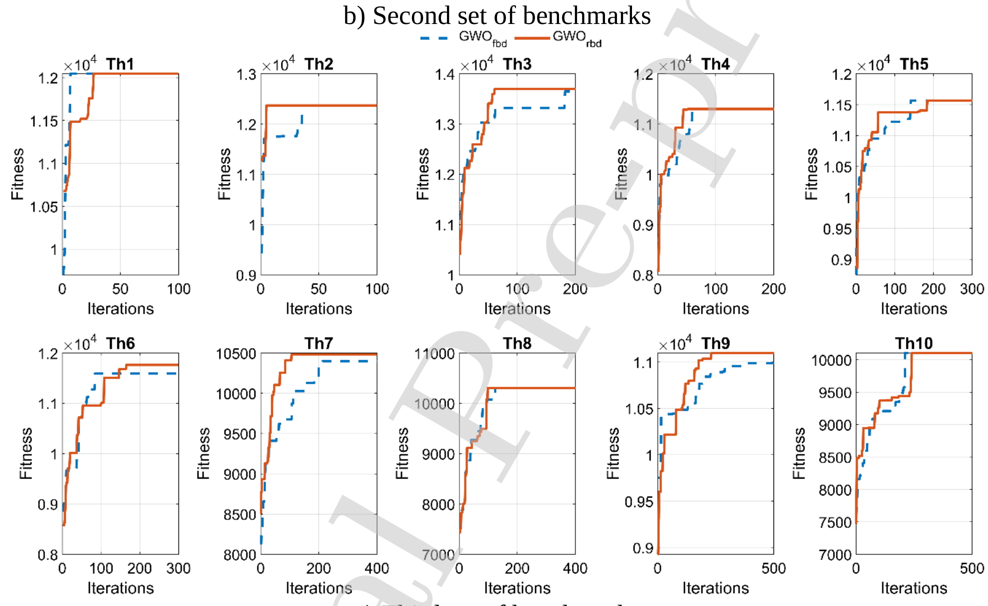
In this section, computational results for the SUKP are given. Convergence plots of the SUKP instances are given in Figure 7a-7c, respectively. All obtained results are tabulated in Tables 4-5 in which columns represent the instance type, the performance indicators, the compared algorithms, respectively. The results of the compared algorithms are obtained from [40, 41, 52]. The best results are indicated by bold characters.



a) First set of benchmarks



b) Second set of benchmarks



c) Third set of benchmarks

Fig. 7. Convergence plots of the SUKP

Table 4 reports the SUKP solutions for the first set of benchmarks that the number of items is greater than the number of elements. According to results, proposed algorithms can find the best solutions to the compared algorithms for 9 out of 10 problems. In the 5 out of 10 instances, the proposed algorithms have found better results than the compared algorithms. In the instance of Fi5 (300_285_0.10_0.75), GWO$_{fbd}$ outranks the compared algorithm in terms of the best results. Also, GWO$_{rbd}$ achieves best results in the Fi6 (300_285_0.15_0.85), Fi8 (400_385_0.15_0.85), and Fi9 (500_485_0.10_0.75). Both the GWO$_{fbd}$ and GWO$_{rbd}$ find the best results in the Fi10 (500_485_0.15_0.85). Only in the instance of Fi3 (200_185_0.10_0.75), the proposed algorithms are outranked by intAgents. Also, note that the CPU times are not provided in the literature; only the produced results are considered as a performance criterion. In the SUKP, the small-scaled problems take an average of 4.14 seconds, and the large-scaled instances take an average of 278.306 seconds to obtain the results.

In Table 5, the results of the second set of benchmarks are given. According to results, GWO$_{fbd}$ has outranked the other algorithms in the Se5 (300_300_0.10_0.75) instance. For the rest of the instances, both the GWO$_{fbd}$ and GWO$_{rbd}$ have found the best-known results, as reported in Table 5. The results indicate the superiority of the proposed algorithms. It can be inferred from the analysis that the efficiency of the algorithms becomes more apparent when the problem size grows. Especially in small-scaled instances, the majority of the algorithms in the literature can find the best-reported results, while medium- and large-scaled instances provide a good indication about the performance of the algorithms.

The results of the third group of instances are reported in Table 6. According to results, GWO$_{rbd}$ has found superior results in the instances of Th6 (285_300_0.15_0.85) and Th9 (485_500_0.10_0.75). Furthermore, proposed algorithms have found the best results reported by the compared algorithms except for the instance Th8 (385_400_0.15_0.85). In a total of 9 instances out of 10, the proposed algorithms become competitive with the reported results. As in the case of second instances, the performance of the proposed algorithms becomes more apparent as the problem size grows.

Table 4. Computational results of the first type of instances for SUKP

instance	*results*	A-SUKP	GA	BABC	ABC$_{bin}$	binDE	bWSA	gPSO*	gPSO	intAgents	Djaya	GWO$_{fbd}$	GWO$_{rbd}$
Fi1: 100_85_0.10_0.75	*best*	12,459	13,044	13,251	13,044	13,044	13,044	13,167	**13,283**	**13,283**	**13,283**	13,089	**13,283**
	mean	12,459.00	12,956.40	13,028.50	12,818.50	12,991.00	12,915.67	12,937.05	13,050.53	13,061.02	13,076.00	13,041.37	13,065.93
	std. dev.	0.00	130.66	92.63	153.06	75.95	185.45	189.60	37.41	44.08	66.61	31.17	70.02
Fi2: 100_85_0.15_0.85	*best*	11,119	12,066	12,238	12,238	12,274	12,238	12,210	**12,274**	**12,274**	**12,274**	**12,274**	**12,274**
	mean	11,119.00	11,546.00	12,155.00	12,049.30	12,123.90	11,527.41	11,777.71	12,084.82	12,074.84	12,192.50	12,079.40	12,053.57
	std. dev.	0.00	214.94	53.29	96.11	67.61	332.27	277.16	95.38	86.37	70.25	93.34	81.99
Fi3: 200_185_0.10_0.75	*best*	11,292	13,064	13,241	12,946	13,241	13,250	13,302	13,405	**13,502**	13,405	13,405	13,405
	mean	11,292.00	12,492.50	13,064.40	11,861.50	12,940.70	12,657.65	12,766.38	13,286.56	13,226.28	13,306.60	13,282.30	13,280.78
	std. dev.	0.00	320.03	99.57	324.65	205.70	319.58	304.82	93.18	150.92	60.96	102.88	123.63
Fi4: 200_185_0.15_0.85	*best*	12,262	13,671	13,829	13,671	13,671	13,858	13,993	14,044	14,044	**14,215**	**14,215**	**14,215**
	mean	12,262.00	12,802.90	13,359.20	12,537.00	13,110.00	12,585.35	12,949.05	13,492.60	13,441.06	13,660.20	13,464.35	13,479.99
	std. dev.	0.00	291.66	234.99	289.53	269.69	302.66	325.58	328.72	324.96	274.76	358.97	358.56
Fi5: 300_285_0.10_0.75	*best*	8,941	10,553	10,428	9,751	10,420	10,991	10,600	11,335	11,335	10,934	**11,413**	11,335
	mean	8,941.00	9,980.87	9,994.76	9,339.30	9,899.24	10,366.21	10,090.47	10,669.51	10,576.10	10,703.20	10,707.54	10,684.17
	std. dev.	0.00	142.97	154.03	158.15	153.18	257.10	236.14	227.85	281.13	112.95	230.46	242.43
Fi6: 300_285_0.15_0.85	*best*	9,432	11,016	12,012	10,913	11,661	12,093	11,935	12,245	12,247	12,245	12,402	**12,259**
	mean	9,432.00	10,349.80	10,902.90	9,957.85	10,499.40	10,901.59	10,750.30	11,607.10	11,490.26	12,037.50	11,646.23	11,606.32
	std. dev.	0.00	215.13	449.45	276.90	403.95	508.79	524.53	477.80	518.81	296.02	517.63	492.99
Fi7: 400_385_0.10_0.75	*best*	9,076	10,083	10,766	9,674	10,576	11,321	10,698	**11,484**	**11,484**	11,337	**11,484**	**11,484**
	mean	9,076.00	9,641.85	10,065.20	9,187.76	9,681.46	10,785.74	9,946.96	10,915.87	10,734.62	11,062.00	10,884.49	10,880.22
	std. dev.	0.00	168.94	241.45	167.08	275.05	361.45	295.28	367.75	371.37	273.63	396.92	386.79
Fi8: 400_385_0.15_0.85	*best*	8,514	9,831	9,649	8,978	9,649	10,435	10,168	10,710	10,710	10,431	10,710	**10,757**
	mean	8,514.00	9,326.77	9,135.98	8,539.95	9,020.87	9,587.72	9,417.20	9,864.55	9,735.00	10,017.90	9,894.54	9,900.01
	std. dev.	0.00	192.20	151.90	161.83	150.99	360.29	360.03	315.38	370.44	207.98	329.34	325.03
Fi9: 500_485_0.10_0.75	*best*	9,864	11,031	10,784	10,340	10,586	11,540	11,258	11,722	11,722	11,722	11,722	**11,771**
	mean	9,864.00	10,567.90	10,452.20	9,910.32	10,363.80	10,921.58	10,565.90	11,184.51	11,111.63	11,269.40	11,276.49	11,338.26
	std. dev.	0.00	123.15	114.35	120.82	93.39	351.69	260.32	322.98	355.18	275.37	347.99	351.46
Fi10: 500_485_0.15_0.85	*best*	8,299	9,472	9,090	8,759	9,191	9,681	9,756	10,022	10,059	9,770	**10,194**	**10,194**
	mean	8,299.00	8,692.67	8,857.89	8,365.04	8,783.99	9,013.09	8,779.44	9,299.56	9,165.26	9,354.28	9,339.80	9,398.07
	std. dev.	0.00	180.12	94.55	114.10	131.05	204.85	300.11	277.62	282.55	212.69	252.98	266.46

Table 5. Computational results of the second type of instances for SUKP

instance	*results*	A-SUKP	GA	BABC	ABC$_{bin}$	binDE	bWSA	gPSO*	gPSO	intAgents	Djaya	GWO$_{fbd}$	GWO$_{rbd}$
Se1: 100_100_0.10_0.75	*best*	13,634	14,044	13,860	13,860	13,814	**14,044**	13,963	**14,044**	**14,044**	**14,044**	**14,044**	**14,044**
	mean	13,634.00	13,806.00	13,734.90	13,547.20	13,675.90	13,492.71	13,739.71	13,854.71	13,767.23	13,912.50	13,861.35	13,847.86
	std. dev.	0.00	144.91	70.76	119.11	119.53	325.34	119.52	96.23	131.59	84.55	84.62	100.33
Se2: 100_100_0.15_0.85	*best*	11,325	13,145	13,508	13,498	13,407	13,407	13,498	**13,508**	**13,508**	**13,508**	**13,508**	**13,508**
	mean	11,325.00	12,234.80	13,352.40	13,103.10	13,212.80	12,487.88	12,937.53	13,347.58	13,003.62	13,439.10	13,312.57	13,297.37
	std. dev.	0.00	388.66	155.14	343.46	287.45	718.23	417.91	194.34	375.74	44.86	189.12	172.16
Se3: 200_200_0.10_0.75	*best*	10,328	11,656	11,846	11,191	11,535	12,271	11,972	**12,522**	**12,522**	**12,522**	12,350	**12,522**
	mean	10,328.00	10,888.70	11,194.30	10,424.10	10,969.40	11,430.23	11,232.55	11,898.73	11,586.26	12,171.60	11,852.44	11,906.97
	std. dev.	0.00	237.85	249.58	197.88	302.52	403.33	349.39	391.83	419.09	220.68	371.57	382.91
Se4: 200_200_0.15_0.85	*best*	9,784	11,792	11,521	11,287	11,469	11,804	12,167	**12,317**	11,911	12,187	11,993	**12,317**
	mean	9,784.00	10,827.50	10,945.00	10,345.90	10,717.10	11,062.06	11,026.81	11,584.64	11,288.25	11,746.00	11,612.07	11,594.90
	std. dev.	0.00	334.43	255.14	273.47	341.08	423.90	421.22	275.32	410.54	181.18	217.13	301.10
Se5: 300_300_0.10_0.75	*best*	10,208	12,055	12,186	11,494	12,304	12,644	12,736	12,695	12,695	12,695	**12,784**	12,695
	mean	10,208.00	11,755.10	11,945.80	10,922.30	11,864.40	12,227.56	11,934.64	12,411.27	12,310.19	12,569.30	12,441.00	12,446.21
	std. dev.	0.00	144.45	127.80	182.63	160.42	308.11	293.83	225.80	238.32	114.13	247.67	227.47
Se6: 300_300_0.15_0.85	*best*	9,183	10,666	10,382	9,633	10,382	11,113	10,724	**11,425**	**11,425**	11,113	**11,425**	**11,425**
	mean	9,183.00	10,099.20	9,859.69	9,186.87	9,710.37	10,216.71	9,906.81	10,568.41	10,384.00	10,701.90	10,632.71	10,648.53
	std. dev.	0.00	337.42	177.02	147.78	208.48	351.12	399.13	327.48	378.42	153.66	345.63	328.13
Se7: 400_400_0.10_0.75	*best*	9,751	10,570	10,626	10,160	10,462	11,199	11,048	**11,531**	**11,531**	11,310	**11,531**	**11,531**
	mean	9,751.00	10,112.40	10,101.10	9,549.04	9,975.80	10,624.79	10,399.97	10,958.96	10,756.92	10,914.80	10,961.25	10,964.98
	std. dev.	0.00	157.89	196.99	141.27	185.57	266.46	281.99	274.90	250.56	216.47	258.47	276.16
Se8: 400_400_0.15_0.85	*best*	8,497	9,235	9,541	9,033	9,388	10,915	10,264	**10,927**	**10,927**	10,915	**10,927**	**10,927**
	mean	8,497.00	8,793.76	9,032.95	8,365.62	8,768.42	9,580.64	9,195.24	9,845.17	9,608.07	9,969.90	9,849.04	9,873.27
	std. dev.	0.00	169.52	194.18	153.40	212.24	411.83	311.90	358.91	363.72	287.61	343.90	373.70
Se9: 500_500_0.10_0.75	*best*	9,615	10,460	10,755	10,071	10,546	10,827	10,647	10,888	**10,960**	**10,960**	10,921	**10,960**
	mean	9,615.00	10,185.40	10,328.50	9,738.17	10,227.70	10,482.80	10,205.08	10,681.46	10,610.53	10,703.50	10,716.55	10,742.98
	std. dev.	0.00	114.19	91.62	111.63	103.32	165.62	190.05	125.36	169.73	105.18	140.87	130.05
Se10: 500_500_0.15_0.85	*best*	7,883	9,496	9,318	9,262	9,312	10,082	9,839	**10,194**	10,381	10,176	**10,194**	**10,194**
	mean	7,883.00	8,882.88	9,180.74	8,617.91	9,096.13	9,478.71	9,106.64	9,703.62	9,578.89	9,801.50	9,758.61	9,737.48
	std. dev.	0.00	158.21	84.91	141.32	145.45	262.44	257.65	252.84	278.06	222.21	243.59	272.51

Table 6. Computational results of the third type of instances for SUKP

instance	*results*	A-SUKP	GA	BABC	ABC_{bin}	binDE	bWSA	gPSO*	gPSO	intAgents	Djaya	GWO_{fbd}	GWO_{rbd}
Th1: 85_100_0.10_0.75	*best*	10,231	11,454	11,664	11,206	11,352	11,947	11,710	**12,045**	**12,045**	**12,045**	**12,045**	**12,045**
	mean	10,231.00	11,092.70	11,182.70	10,879.50	11,075.00	11,233.16	11,237.05	11,486.95	11,419.75	11,570.60	11,441.23	11,430.44
	std. dev.	0.00	171.22	183.57	163.62	119.42	216.67	168.96	137.52	140.77	177.86	111,72	127.56
Th2: 85_100_0.15_0.85	*best*	10,483	12,124	12,369	12,006	**12,369**	**12,369**	**12,369**	**12,369**	**12,369**	**12,369**	**12,369**	**12,369**
	mean	10,483.00	11,326.30	12,081.60	11,485.30	11,875.90	11,342.70	11,684.46	11,994.36	11,885.21	12,318.00	11,917,83	11,942.93
	std. dev.	0.00	417.00	193.79	248.33	336.94	474.76	353.79	436.81	431.67	181.92	442.25	418.72
Th3: 185_200_0.10_0.75	*best*	11,508	12,841	13,047	12,308	13,024	13,505	13,298	**13,696**	**13,696**	**13,696**	13,647	**13,696**
	mean	11,508.00	12,236.60	12,522.80	11,667.90	12,277.50	12,689.09	12,514.72	13,204.26	13,084.52	13,350.20	13,121.23	13,125.85
	std. dev.	0.00	198.18	201.35	177.14	234.24	336.51	356.20	366.56	388.39	182.56	365.41	367.06
Th4: 185_200_0.15_0.85	*best*	8,621	10,920	10,602	10,376	10,547	10,831	10,856	**11,298**	**11,298**	**11,298**	**11,298**	**11,298**
	mean	8,621.00	10,351.50	10,150.60	9,684.33	10,085.40	10,228.07	10,208.33	10,801.41	10,780.14	10,828.90	10,871.49	10,819.34
	std. dev.	0.00	208.08	152.91	184.84	160.60	286.92	263.73	205.76	239.61	191.76	240.32	239.52
Th5: 285_300_0.10_0.75	*best*	9,961	10,994	11,158	10,269	11,152	**11,538**	11,310	**11,568**	**11,568**	**11,568**	**11,568**	**11,568**
	mean	9,961.00	10,640.10	10,775.90	9,957.09	10,661.30	11,105.09	10,761.96	11,317.99	11,205.72	11,327.70	10,001.33	10,014.47
	std. dev.	0.00	126.84	116.80	141.48	149.84	197.78	199.43	182.82	258.49	166.91	174.62	215.35
Th6: 285_300_0.15_0.85	*best*	9,618	11,093	10,528	10,051	10,528	11,377	11,226	11,517	11,517	11,401	11,590	**11,763**
	mean	9,618.00	10,190.30	9,897.92	9,424.15	9,832.32	10,452.03	10,309.19	10,899.20	10,747.33	11,025.90	10,470.33	10,871.49
	std. dev.	0.00	249.76	186.53	197.14	232.72	416.76	389.12	300,36	334.25	208.08	340.47	332.09
Th7: 385_400_0.10_0.75	*best*	8,672	9,799	10,085	9,235	9,883	10,414	9,871	**10,483**	10,326	10,414	10,397	**10,483**
	mean	8,672.00	9,432.82	9,537.50	8,904.94	9,314.57	9,778.03	9,552.14	10,013.43	9,892.17	10,017.00	10,043.23	9,902.72
	std. dev.	0.00	163.84	184.62	111.85	191.59	221.49	234.10	202.40	179.19	141.15	163.95	180.32
Th8: 385_400_0.15_0.85	*best*	8,064	9,173	9,456	8,932	9,352	10,077	9,389	**10,338**	10,131	10,302	10,302	10,302
	mean	8,064.00	8,703.66	9,090.03	8,407.06	8,846.99	9,203.52	8,881.17	9,524.98	9,339.67	9,565.72	9,472.39	9,455.24
	std. dev.	0.00	154.15	156.69	148.52	210.91	303.12	283.30	286.16	288.88	237.90	242.25	261.75
Th9: 485_500_0.10_0.75	*best*	9,559	10,311	10,823	10,357	10,728	10,835	10,595	11,094	11,094	10,971	10,989	**11,097**
	mean	9,559.00	9,993.16	10,483.40	9,615.37	10,159.40	10,607.21	10,145.26	10,687.62	10,603.53	10,754.80	10,702.72	10,725.26
	std. dev.	0.00	117.73	228.34	151.41	198.49	191.86	199.99	168.06	204.99	112.69	154.92	160.001
Th10: 485_500_0.15_0.85	*best*	8,157	9,329	9,333	8,799	9,218	9,603	9,807	**10,104**	**10,104**	9,715	**10,104**	**10,104**
	mean	8,157.00	8,849.46	9,085.57	8,347.82	8,919.64	9,141.94	8,917.44	9,383.28	9,259.36	9,467.80	9,462	9,455.24
	std. dev.	0.00	141.84	115.62	122.65	168.90	180.42	267.49	241.01	268.33	106.55	229.88	261.74

Analysis of the mean and standard deviations reveals that the proposed algorithms have shown competitive performance in comparison with the compared algorithms. Although the standard deviations are greater than those of the A-SUKP, GA, BABC, $ABC_{bin,}$ binDE, and bWSA, the proposed algorithms have found superior results when compared with these algorithms. The proposed algorithms have shown similar performance in mean and standard deviations in comparison with the gPSO and intAgents. DJaya has shown superior performance in terms of finding smaller standard deviations. The overall performance of the proposed algorithms shows that both GWO_{fbd} and GWO_{rbd} have competitive performance in all the metrics.

4.3 Computational results for UFLP

In this section, computational results related to UFLP are given. The convergence plots of the UFLP instances are visualized in Fig. 8. The results obtained for the UFLP are given in Table 7. The results of the compared algorithms are obtained from [52]. In some of the algorithms, mean values were not provided in the original articles. In these cases, mean values are represented by *Na*. It can be clearly seen that all of the reported algorithms manifest similar performances, especially in small-scaled instances. It is hard to decide on the superior algorithm by comparing the results. In general, hit values are used to test the performance of the algorithms in the UFLP instances. According to results, the proposed GWO algorithms have shown superior performance in terms of hit values. In the instance of cap132, intAgents algorithm outranked the proposed algorithms. However, the difference is only one hit value, so that the performance of the algorithms can be said to be quite similar. In the literature, the CPU times are not provided, and the hit values are considered as a performance criterion. In the UFLP, the small-scaled problems take an average of 1.4795 seconds, and the large-scaled instances take an average of 8.3261 seconds to obtain the results.

The mean and standard deviation results show that the proposed algorithms have produced high-quality solutions. This is especially visible in the medium- and large-scaled instances. The standard deviations of the proposed algorithm are quite small and even zero for most of the test problems, which shows the robustness of the proposed algorithms. On the other hand, the mean results are also highly competitive.

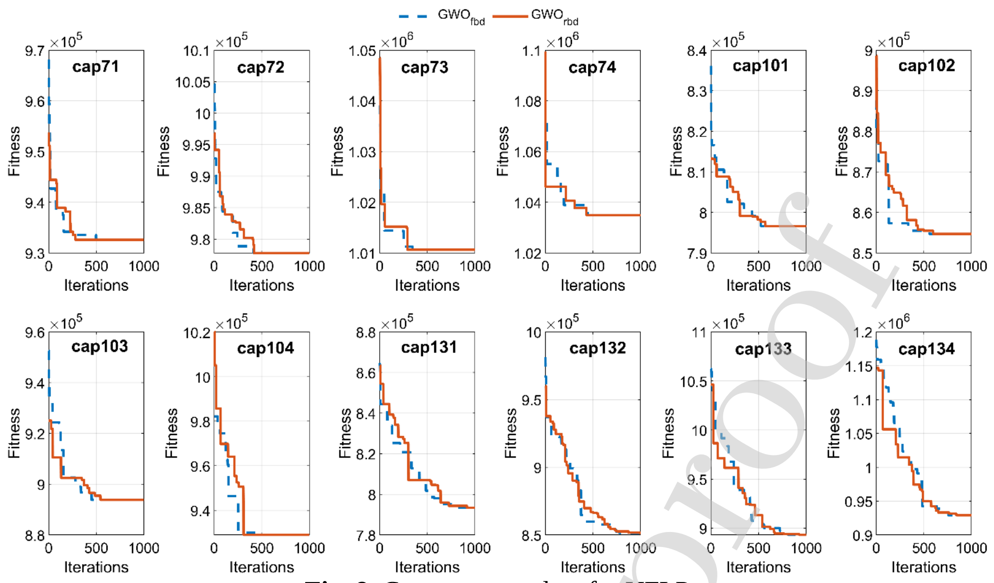


Fig. 8. Convergence plots for UFLP

4.4 Statistical tests

In the experimental studies, it is hard to satisfy the required assumptions of the parametric tests such as independence, normality, and homoscedasticity. In order to perform rigorous comparisons among algorithms, nonparametric tests offer valuable insights into the situations that required assumptions are not validated. In this study, nonparametric tests have been used to compare the algorithms, and the hypothesis tests have been conducted to check statistical significance. In making pairwise comparisons to infer conclusions regarding multiple algorithms, there is a danger of losing control on the family-wise error rate, which is defined as the probability of making false discoveries among all hypotheses [66], statistical tests based on multiple comparisons (1x*N* or *N*x*N* tests) are recommended for comparing multiple algorithms to avoid the problem of uncontrolled family-wise error rate [66].

In this study, the Friedman test with multiple comparisons (*N*x*N*) is used to compare the performance of the algorithms. Once the Friedman test rejects the null hypothesis, the analysis is proceeded with the post-hoc tests to find the differences between algorithms. Nemenyi and Holm tests are used in the post-hoc analysis.

Statistical analysis of the SUKP is given first. The average ranks and the Friedman test results are given in Table 8. The algorithms are ranked from best to worst by assigning ranks from 1 to 12, respectively. Ties are given the average ranks of the corresponding rank sums [66]. According to average ranks for the SUKP, GWO_{rbd} is the best performing algorithm with an average rank of 2.550. On the other hand, GWO_{fbd} ranks fourth, ensuing the intAgents and gPSO algorithms. As a result of the Friedman test, the p-value is 0.000, and the null hypothesis is rejected. In other words, the assumption of the equality of medians is rejected that at least one pair of algorithms has a significant difference.

Table 7. Computational results for UFLP

algorithms		*instances (facilities × demand points)*											
		16×50				25×50				50×50			
		cap71	cap72	cap73	cap74	cap101	cap102	cap103	cap104	cap131	cap132	cap133	cap134
CPSO	*best*	932615.75	977799.40	1010641.45	1034976.98	796648.44	854704.20	893782.11	928941.75	795291.86	851495.33	893076.71	928941.75
	mean	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*
	std.	562.23	1324.30	702.13	2124.54	1480.72	1015.64	1695.79	3842.64	2429.54	4297.07	4210.93	6619.05
	hit	25	25	22	0	0	10	0	18	0	0	0	7
ABC$_{bin}$	*best*	932615.75	977799.40	1010641.45	1034976.98	796648.44	854704.20	893782.11	928941.75	793439.56	851495.33	893076.71	928941.75
	mean	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*
	std.	0.00	0.00	0.00	0.00	0.00	0.00	85.67	0.00	1065.73	213.28	561.34	0.00
	hit	**30**	**30**	**30**	**30**	**30**	**30**	25	**30**	6	14	5	**30**
bWSA	*best*	932615.750	977799.40	1010641.45	1034976.97	796648.43	854704.20	893782.11	928941.75	793439.56	851495.32	893076.71	928941.75
	mean	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*	*Na*
	std.	0.00	0.000	0.00	0.00	380.43	0.00	470.95	0.00	1025.78	251.65	501.91	1016.14
	hit	**30**	**30**	**30**	**30**	22	**30**	10	**30**	6	23	7	26
PSO	*best*	932615.75	977799.40	1010641.45	1034976.97	796648.43	854704.20	893782.11	928941.75	803826.01	879785.46	921608.03	944335.68
	mean	932615.75	977799.40	1010641.45	1034976.97	799006.37	857115.01	896226.99	933746.35	817976.18	889889.93	943216.76	1002804.33
	std.	0.00	0.00	0.00	0.00	1085.48	1268.63	1763.88	3358.73	4545.68	4697.11	9163.94	19102.20
	hit	**30**	**30**	**30**	**30**	1	2	3	2	0	0	0	0
GWO	*best*	932615.75	977799.40	1010641.45	1034976.97	796648.43	854704.20	893782.11	928941.75	793439.56	851495.32	893076.71	928941.75
	mean	932615.75	978056.06	1010702.63	1034976.97	797123.96	854830.95	894083.99	928941.75	794606.70	852034.68	894088.39	929716.82
	std.	0.00	976.76	335.11	0.00	772.61	386.76	480.63	0.00	1292.51	789.94	1006.16	1783.31
	hit	**30**	28	29	**30**	19	27	20	**30**	11	13	5	20
*prLe*GWO	*best*	932615.75	977799.40	1010641.45	1034976.97	796648.43	854704.20	893782.11	928941.75	793439.56	851495.32	893076.71	928941.75
	mean	932615.75	977799.40	1010641.45	1034976.97	796648.43	854704.20	893865.11	928941.75	793781.44	851616.90	893717.00	928941.75
	std.	0.00	0.00	0.00	0.00	0.00	0.00	260.79	0.00	426.88	429.35	571.45	0.00
	hit	**30**	**30**	**30**	**30**	**30**	**30**	26	**30**	18	26	10	**30**
intAgents	*best*	932615.75	977799.40	1010641.45	1034976.97	796648.44	854704.20	893782.11	928941.75	793439.56	851495.33	893076.71	928941.75
	mean	932615.75	977799.40	1010641.45	1034976.97	796763.14	854704.20	893872.65	928941.75	793755.00	851495.32	893541.77	928941.75
	std.	0.00	0.00	0.00	0.00	297.44	0.00	290.35	0.00	421.65	0.00	457.48	0.00
	hit	**30**	**30**	**30**	**30**	26	**30**	26	**30**	19	**30**	13	**30**
GWO$_{fbd}$	*best*	932,615.75	977,799.40	1,010,641.45	1,034,976.97	796,648.44	854,704.20	893,782.11	928,941.75	793,439.56	851,495.33	893,076.71	928,941.75
	mean	932,615.75	977,799.40	1,010,641.45	1,034,976.98	796,648.44	854,704.20	893,782.11	928,941.75	793,520.91	851,544.90	893,345.37	928,941.75
	std.	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	249.12	190.06	337.45	0.00
	hit	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**27**	28	17	**30**
GWO$_{rbd}$	*best*	932,615.75	977,799.40	1,010,641.45	1,034,976.97	796,648.44	854,704.20	893,782.11	928,941.75	793,439.56	851,495.33	893,076.71	928,941.75
	mean	932,615.75	977,799.40	1,010,641.45	1,034,976.98	796,648.44	854,704.20	893,850.05	928,941.75	793,549.58	851,501.15	893,285.29	928,941.75
	std.	0.00	0.00	0.00	0.00	0.00	0.00	258.54	0.00	286.19	31.91	332.73	0.00
	hit	**30**	**30**	**30**	**30**	**30**	**30**	28	**30**	26	29	**20**	**30**

Table 8. Average ranks for SUKP and obtained *p*-values of (Friedman Tests)

algorithms	*overall avg. ranks*
A-SUKP	12.000
ABCbin	10.617
GA	9.083
binDE	9.050
BABC	7.933
gPSO*	6.883
bWSA	6.283
Djaya	4.033
GWO_{fbd}	3.233
intAgents	3.217
gPSO	3.117
GWO_{rbd}	**2.550**
p-values (*adj. ties*)	0.000
decision on H_0	reject H_0

Nemenyi and Holm tests are carried out to find out the algorithms that have significant differences. The z-scores, unadjusted-p values, Nemenyi and Holm test results are given in Table 9. For the sake of readability, pairwise comparisons related to GWO_{fbd} and GWO_{rbd} are given. According to results, significant differences between binary GWO algorithms and A-SUKP, ABCbin, GA, binDE, BABC, gPSO*, bWSA have been discovered. On the other hand, GWO_{fbd} and GWO_{rbd} are not statistically different from each other. Furthermore, the post-hoc analysis states that the GWO_{fbd} and GWO_{rbd} algorithms are as good as state-of-the-art optimizers Djaya, gPSO, and intAgents.

Table 9. Pairwise comparisons and post-hoc test results for SUKP

		best results					*best results*		
comparisons	*z-score*	*unadjusted*	*Nemenyi*	*Holm*	*comparisons*	*z-score*	*unadjusted*	*Nemenyi*	*Holm*
GWO_{rbd} vs A-SUKP	10.151	0.000 (+)	0.000 (+)	0.000 (+)	GWO_{rbd} vs bWSA	4.010	0.000 (+)	0.004 (+)	0.002 (+)
GWO_{rbd} vs ABCbin	8.665	0.000 (+)	0.000 (+)	0.000 (+)	GWO_{fbd} vs gPSO*	3.921	0.000 (+)	0.006 (+)	0.003 (+)
GWO_{fbd} vs A-SUKP	9.417	0.000 (+)	0.000 (+)	0.000 (+)	GWO_{fbd} vs bWSA	3.276	0.001 (+)	0.069 (~)	0.031 (+)
GWO_{fbd} vs ABCbin	7.931	0.000 (+)	0.000 (+)	0.000 (+)	GWO_{rbd} vs Djaya	1.593	0.111 (~)	1.000 (~)	1.000 (~)
GWO_{rbd} vs GA	7.018	0.000 (+)	0.000 (+)	0.000 (+)	GWO_{fbd} vs Djaya	0.859	0.390 (~)	1.000 (~)	1.000 (~)
GWO_{rbd} vs binDE	6.982	0.000 (+)	0.000 (+)	0.000 (+)	GWO_{rbd} vs GWO_{fbd}	0.734	0.463 (~)	1.000 (~)	1.000 (~)
GWO_{fbd} vs GA	6.284	0.000 (+)	0.000 (+)	0.000 (+)	GWO_{rbd} vs intAgents	0.716	0.474 (~)	1.000 (~)	1.000 (~)
GWO_{fbd} vs binDE	6.248	0.000 (+)	0.000 (+)	0.000 (+)	GWO_{rbd} vs gPSO	0.609	0.543 (~)	1.000 (~)	1.000 (~)
GWO_{rbd} vs BABC	5.783	0.000 (+)	0.000 (+)	0.000 (+)	gPSO vs GWO_{fbd}	0.125	0.900 (~)	1.000 (~)	1.000 (~)
GWO_{fbd} vs BABC	5.049	0.000 (+)	0.000 (+)	0.000 (+)	intAgents vs GWO_{fbd}	0.018	0.986 (~)	1.000 (~)	1.000 (~)
GWO_{rbd} vs gPSO*	4.655	0.000 (+)	0.000 (+)	0.000 (+)					

Similarly, the Friedman test has been performed for the UFLP based on hit values. The average ranks for the UFLP are given in Table 10. According to results, GWO_{rbd} and GWO_{fbd} have an average rank of 3.08 that is sharing the first rank. The resulting p-value is found to be 0.000; hence, the null hypothesis is rejected. In other words, medians are not the same, and at least one pair of algorithms are statistically different from each other.

To determine which algorithms are statistically different from each other, Nemenyi and Holm post-hoc procedures are implemented. According to results given in Table 11, the proposed algorithms are statistically different from CPSO and PSO algorithms. Additionally, it has been statistically proven that the proposed algorithms are at least as good as the state-of-the-art binary optimization algorithms such as bWSA, ABCBin, prLeGWO, and intAgents.

Table 10. Average ranks for UFLP and obtained *p*-values of (Friedman Tests)

algorithms	*overall avg. ranks*
CPSO	8.63
PSO	7.13

GWO	6.21
bWSA	5.00
ABCBin	4.50
prLeGWO	3.79
intAgents	3.58
GWOfbd	3.08
GWOrbd	**3.08**
p-values (*adj. ties*)	0.000
decision on H_0	reject H_0

Table 11. Pairwise comparisons and post-hoc test results for UFLP

			best results					*best results*	
comparisons	*z-score*	*unadjusted*	*Nemenyi*	*Holm*	*comparisons*	*z-score*	*unadjusted*	*Nemenyi*	*Holm*
GWO$_{rbd}$ vs CPSO	4.957	0.000 (+)	0.000 (+)	0.000 (+)	GWO$_{rbd}$ vs ABCBin	1.267	0.205 (~)	1.000 (~)	1.000 (~)
GWO$_{fbd}$ vs CPSO	4.957	0.000 (+)	0.000 (+)	0.000 (+)	GWO$_{fbd}$ vs ABCBin	1.267	0.205 (~)	1.000 (~)	1.000 (~)
GWO$_{rbd}$ vs PSO	3.615	0.000 (+)	0.011 (+)	0.009 (+)	GWO$_{rbd}$ vs prLeGWO	0.634	0.526 (~)	1.000 (~)	1.000 (~)
GWO$_{fbd}$ vs PSO	3.615	0.000 (+)	0.011 (+)	0.009 (+)	GWO$_{fbd}$ vs prLeGWO	0.634	0.526 (~)	1.000 (~)	1.000 (~)
GWO$_{rbd}$ vs GWO	2.795	0.005 (+)	0.187 (~)	0.135 (~)	GWO$_{rbd}$ vs intAgents	0.447	0.655 (~)	1.000 (~)	1.000 (~)
GWO$_{fbd}$ vs GWO	2.795	0.005 (+)	0.187 (~)	0.135 (~)	GWO$_{fbd}$ vs intAgents	0.447	0.655 (~)	1.000 (~)	1.000 (~)
GWO$_{rbd}$ vs bWSA	1.714	0.086 (~)	1.000 (~)	1.000 (~)	GWO$_{rbd}$ vs GWOfbd	0.000	1.000 (~)	1.000 (~)	1.000 (~)
GWO$_{fbd}$ vs bWSA	1.714	0.086 (~)	1.000 (~)	1.000 (~)					

Finally, it can be concluded that the GWO$_{rbd}$ and GWO$_{fbd}$ are powerful binary optimization algorithms, producing highly competitive results with the state-of-the-art solvers. Statistical analysis demonstrated that both the GWO$_{rbd}$ and GWO$_{fbd}$ have a statistical difference with the recently proposed binary optimization algorithms. For the rest of the algorithms, the proposed algorithms yield the best-found solutions of the other algorithms as well.

4.5 The effects of the crossover and mutation operations

In this section, component-based analysis has been performed in order to verify the effectiveness of MP-XOver and adaptive mutation operations. To this end, canonical GWO algorithm with sigmoid transfer function (GWO$_{sigmoid}$), binary GWO algorithm with only adaptive mutation operation (GWO$_{Mutation}$), binary GWO algorithm with fitness-based selection and MP-Xover operation (GWO$_{Crossover\_fbd}$), binary GWO algorithm with rank-based selection and MP-Xover operation (GWO$_{Crossover\_rbd}$), and proposed algorithms (GWO$_{fbd}$ and GWO$_{rbd}$) are used to solve benchmark suites of SUKP and UFLP, and the results are statistically verified. Note that GWO$_{Crossover\_fbd}$ and GWO$_{Crossover\_rbd}$ do not involve mutation operation so that the effects of different algorithm components are analyzed. 10 instances of SUKP (m=n) and 12 instances of the UFLP problem are solved by using 6 different algorithms. Table 12-13 shows the results of the SUKP and UFLP, respectively.

Table 12. Effect of algorithm components for SUKP

instance	*results*	GWO$_{Sigmoid}$	GWO$_{Mutation}$	GWO$_{Crossover\_fbd}$	GWO$_{Crossover\_rbd}$	GWO$_{fbd}$	GWO$_{rbd}$
100_100_0.10_0.75	*best*	13,767	13,957	13,849	13,908	**14,044**	**14,044**
	mean	13,504.73	13,685.67	13,645.91	13,604.30	13,861.35	13,847.86
	std. dev.	158.73	116.50	161.18	187.91	84.62	100.33
100_100_0.15_0.85	*best*	13,407	13,407	13,407	13,407.00	**13,508**	**13,508**
	mean	12,909.07	12,772.47	12,583.36	12,526.50	13,312.57	13,297.37
	std. dev.	362.89	361.60	476.49	556.89	189.12	172.16
200_200_0.10_0.75	*best*	10,909	11,663	11,553	11,395	12,350	**12,522**
	mean	10,354.63	10,745.25	10,589.06	10,482.26	11,852.44	11,906.97
	std. dev.	251.06	231.76	392.52	379.13	371.57	382.91
200_200_0.15_0.85	*best*	11,072	11,320	11,800	11,738	11,993	**12,317**
	mean	10,271.73	10,480.20	10,465.98	10,388.61	11,612.07	11,594.90
	std. dev.	320.20	238.46	547.41	501.01	217.13	301.1
300_300_0.10_0.75	*best*	11,173	12,332	11,893	11,864	**12,784**	12,695

	mean	10,723.73	11,372.93	10,996.20	11,063.15	12,441.00	12,446.21
	std. dev.	195.65	211.88	397.60	350.78	247.67	227.47
300_300_0.15_0.85	*best*	9,501	10,136	10,359	9,928	**11,425**	**11,425**
	mean	9,107.67	9,453.25	9,120.48	9,073.47	10,632.71	10,648.53
	std. dev.	189.84	209.14	333.54	299.97	345.63	328.13
400_400_0.10_0.75	*best*	10,093	10,509	10,283	10,161	**11,531**	**11,531**
	mean	9,535.23	9,820.25	9,578.81	9,540.76	10,961.25	10,964.98
	std. dev.	173.78	164.97	276.38	254.22	258.47	276.16
400_400_0.15_0.85	*best*	8,780	9,089	9,013	9,094	**10,927**	**10,927**
	mean	8,335.40	8,538.66	8,204.72	8,197.32	9,849.04	9,873.27
	std. dev.	172.80	183.87	320.33	302.57	343.9	373.7
500_500_0.10_0.75	*best*	10,027	10,314	10,059	10,180	10,921	**10,960**
	mean	9,633.50	9,913.07	9,628.87	9,600.54	10,716.55	10,742.98
	std. dev.	122.69	120.71	222.89	240.65	140.87	130.05
500_500_0.15_0.85	*best*	8,904	9,036	9,248	9,037	**10,194**	**10,194**
	mean	8,543.93	8,702.69	8,437.29	8,396.06	9,758.61	9,737.48
	std. dev.	126.08	145.39	264.28	294.02	243.59	272.51

Table 13. Effect of algorithm components for UFLP

instance	*results*	GWO$_{Sigmoid}$	GWO$_{Mutation}$	GWO$_{Crossover\_fbd}$	GWO$_{Crossover\_rbd}$	GWO$_{fbd}$	GWO$_{rbd}$
cap71	*best*	932,615.75	932,615.75	932,615.75	932,615.75	932,615.75	932,615.75
	mean	934,104.20	934,554.77	937,150.33	938,741.22	932,615.75	932,615.75
	std. dev.	1,503.78	1,203.96	4,123.47	5,041.35	0	0
	hit	16	3	5	5	**30**	**30**
cap72	*best*	977,799.40	977,799.40	977,799.40	977,799.40	977,799.40	977,799.40
	mean	979,380.23	979,558.96	986,289.19	984,427.33	977,799.40	977,799.40
	std. dev.	1,645.21	1,475.17	6,374.35	4,727.21	0	0
	hit	17	8	1	3	**30**	**30**
cap73	*best*	1,010,641.45	1,010,641.45	1,011,234.36	1,010,641.45	1,010,641.45	1,010,641.45
	mean	1,011,564.67	1,011,902.65	1,017,506.31	1,017,704.07	1,010,641.45	1,010,641.45
	std. dev.	1,303.77	1,309.65	4,949.55	5,387.97	0	0
	hit	17	5	1	2	**30**	**30**
cap74	*best*	1,034,976.98	1,034,976.98	1,034,976.98	1,034,976.98	1,034,976.97	1,034,976.97
	mean	1,038,627.75	1,039,057.97	1,050,103.51	1,047,705.18	1,034,976.98	1,034,976.98
	std. dev.	3,171.09	3,566.46	12,209.38	13,338.13	0	0
	hit	15	8	4	6	**30**	**30**
cap101	*best*	796,648.44	800,130.31	797,508.73	798,610.43	796,648.44	796,648.44
	mean	803,125.23	804,292.26	804,867.44	804,676.88	796,648.44	796,648.44
	std. dev.	2,257.36	2,237.27	3,972.26	5,267.68	0	0
	hit	1	1	1	1	**30**	**30**
cap102	*best*	855,971.75	858,026.70	856,734.40	854,704.20	854,704.20	854,704.20
	mean	861,660.99	863,988.80	864,586.95	864,524.72	854,704.20	854,704.20
	std. dev.	2,618.15	2,373.60	5,612.43	7,161.76	0	0
	hit	1	1	1	1	**30**	**30**
cap103	*best*	898,149.25	897,532.49	893,782.11	893,782.11	893,782.11	893,782.11
	mean	903,577.02	905,781.16	902,361.11	903,893.56	893,782.11	893,850.05
	std. dev.	3,484.23	4,516.13	7,596.14	8,338.10	0	258.54
	hit	1	1	1	1	**30**	28
cap104	*best*	932,985.33	938,737.99	928,941.75	928,941.75	928,941.75	928,941.75
	mean	951,858.85	956,184.46	951,612.19	949,425.21	928,941.75	928,941.75
	std. dev.	9,394.91	7,917.76	16,882.37	15,492.45	0	0
	hit	1	1	2	2	**30**	**30**
cap131	*best*	817,109.79	826,869.49	796,559.94	802,475.11	793,439.56	793,439.56
	mean	828,420.12	833,519.86	809,263.87	811,510.75	793,520.91	793,549.58
	std. dev.	5,496.14	3,601.98	6,928.67	7,806.21	249.12	286.19
	hit	1	1	1	1	**27**	26

cap132	*best*	891,758.86	895,345.68	853,171.53	857,555.14	851,495.33	851,495.33
	mean	908,884.42	914,279.77	873,856.51	876,832.74	851,544.90	851,501.15
	std. dev.	7,667.99	7,974.58	10,476.89	11,066.19	190.06	31.91
	hit	1	1	1	1	28	**29**
cap133	*best*	939,619.26	957,864.70	898,236.65	894,530.31	893,076.71	893,076.71
	mean	973,535.98	985,155.44	921,308.45	923,505.96	893,345.37	893,285.29
	std. dev.	13,009.85	10,701.49	13,898.00	13,626.29	337.45	332.73
	hit	1	1	1	1	17	**20**
cap134	*best*	996,694.15	1,018,929.59	940,218.99	931,507.55	928,941.75	928,941.75
	mean	1,065,330.13	1,085,044.36	969,442.76	976,247.72	928,941.75	928,941.75
	std. dev.	24,882.64	18,726.83	22,038.75	27,122.40	0	0
	hit	1	1	1	1	**30**	**30**

According to results, the proposed algorithms outperform the other variants for both of the SUKP and UFLP. In the SUKP, GWO$_{\text{fbd}}$ and GWO$_{\text{rbd}}$ outrank GWO$_{\text{Sigmoid}}$, GWO$_{\text{Mutation}}$, GWO$_{\text{Crossover\_fbd}}$, and GWO$_{\text{Crossover\_rbd}}$. Another observation is that the use of transfer function within the GWO algorithm results in poor performance for the SUKP problem. Algorithms that directly operate on the binary solution vector exhibit better performance than a transfer function-based approach. On the other hand, the findings of UFLP clearly show that the hit values of GWO$_{\text{Sigmoid}}$, GWO$_{\text{Mutation}}$, GWO$_{\text{Crossover\_fbd}}$, and GWO$_{\text{Crossover\_rbd}}$ are quite low, which indicates that the algorithms cannot yield robust results. Moreover, in the UFLP problem, applying only crossover and mutation operations exhibits poor performance.

To further clarify the effect of algorithm components on the overall performance, statistical verification has been conducted. According to the Friedman test, the p-value for the SUKP is 0.000, which shows that the performances of the algorithms are not identical. To detect which algorithms are statistically different from each other, post-hoc analyses with Nemenyi and Holm tests are performed. As can be seen from Table 14, GWO$_{\text{fbd}}$ and GWO$_{\text{rbd}}$ are statistically different from the other algorithms. Furthermore, post-hoc analysis shows that the algorithms, which make use of only crossover, mutation, and transfer function, cannot outrank each other and exhibit similar performance.

Table 14. Statistical comparison of binary GWO variants for SUKP

comparisons	*z-score*	*unadjusted*	*Nemenyi*	*Holm*
GWO$_{\text{rbd}}$ vs GWO$_{\text{Sigmoid}}$	5.319	0 (+)	0 (+)	0 (+)
GWO$_{\text{fbd}}$ vs GWO$_{\text{Sigmoid}}$	5.08	0 (+)	0 (+)	0 (+)
GWO$_{\text{rbd}}$ vs GWO$_{\text{Crossover\_rbd}}$	3.526	0 (+)	0.006 (+)	0.005 (+)
GWO$_{\text{fbd}}$ vs GWO$_{\text{Crossover\_rbd}}$	3.287	0.001 (+)	0.015 (+)	0.012 (+)
GWO$_{\text{rbd}}$ vs GWO$_{\text{Crossover\_fbd}}$	3.167	0.002 (+)	0.023 (+)	0.017 (+)
GWO$_{\text{fbd}}$ vs GWO$_{\text{Crossover\_fbd}}$	2.928	0.003 (+)	0.051 (~)	0.034 (+)
GWO$_{\text{rbd}}$ vs GWO$_{\text{Mutation}}$	2.809	0.005 (+)	0.075 (~)	0.045 (+)
GWO$_{\text{fbd}}$ vs GWO$_{\text{Mutation}}$	2.57	0.01 (+)	0.153 (~)	0.081 (~)
GWO$_{\text{Mutation}}$ vs GWO$_{\text{Sigmoid}}$	2.51	0.012 (+)	0.181 (~)	0.085 (~)
GWO$_{\text{Crossover\_fbd}}$ vs GWO$_{\text{Sigmoid}}$	2.151	0.031 (+)	0.472 (~)	0.189 (~)
GWO$_{\text{Crossover\_rbd}}$ vs GWO$_{\text{Sigmoid}}$	1.793	0.073 (~)	1 (~)	0.365 (~)
GWO$_{\text{Mutation}}$ vs GWO$_{\text{Crossover\_rbd}}$	0.717	0.473 (~)	1 (~)	1 (~)
GWO$_{\text{Mutation}}$ vs GWO$_{\text{Crossover\_fbd}}$	0.359	0.72 (~)	1 (~)	1 (~)
GWO$_{\text{Crossover\_fbd}}$ vs GWO$_{\text{Crossover\_rbd}}$	0.359	0.72 (~)	1 (~)	1 (~)
GWO$_{\text{rbd}}$ vs GWO$_{\text{fbd}}$	0.239	0.811 (~)	1 (~)	1 (~)

In analogous with the SUKP results, a statistical significance has been detected between the performance of the proposed methods and GWO$_{\text{Sigmoid}}$, GWO$_{\text{Mutation}}$, GWO$_{\text{Crossover\_fbd}}$, and GWO$_{\text{Crossover\_rbd}}$. The Friedman test computes the p-value of 0.000. Table 15 shows that both

Nemenyi and Holm tests indicate statistical differences between proposed algorithms and the compared variants of the GWO. According to post-hoc analysis, both GWO$_{rbd}$ and GWO$_{fbd}$ are statistically different from GWO$_{Sigmoid}$, GWO$_{Crossover\_fbd}$, GWO$_{Crossover\_rbd}$, and GWO$_{Mutation}$. On the other hand, GWO$_{rbd}$ and GWO$_{fbd}$ cannot be distinguished in terms of statistical significance.

Table 15. Statistical comparison of binary GWO variants for UFLP

comparisons	*z-score*	*unadjusted*	*Nemenyi*	*Holm*
GWO$_{Crossover\_fbd}$ vs GWO$_{fbd}$	4.31	0 (+)	0 (+)	0 (+)
GWO$_{Crossover\_fbd}$ vs GWO$_{rbd}$	4.31	0 (+)	0 (+)	0 (+)
GWO$_{Mutation}$ vs GWO$_{fbd}$	4.037	0 (+)	0.001 (+)	0.001 (+)
GWO$_{Mutation}$ vs GWO$_{rbd}$	4.037	0 (+)	0.001 (+)	0.001 (+)
GWO$_{Crossover\_rbd}$ vs GWO$_{fbd}$	3.982	0 (+)	0.001 (+)	0.001 (+)
GWO$_{Crossover\_rbd}$ vs GWO$_{rbd}$	3.982	0 (+)	0.001 (+)	0.001 (+)
GWO$_{Sigmoid}$ vs GWO$_{fbd}$	3.382	0.001 (+)	0.011 (+)	0.006 (+)
GWO$_{Sigmoid}$ vs GWO$_{rbd}$	3.382	0.001 (+)	0.011 (+)	0.006 (+)
GWO$_{Crossover\_fbd}$ vs GWO$_{Sigmoid}$	0.927	0.354 (~)	1 (~)	1 (~)
GWO$_{Mutation}$ vs GWO$_{Sigmoid}$	0.655	0.513 (~)	1 (~)	1 (~)
GWO$_{Crossover\_rbd}$ vs GWO$_{Sigmoid}$	0.6	0.548 (~)	1 (~)	1 (~)
GWO$_{Crossover\_fbd}$ vs GWO$_{Crossover\_rbd}$	0.327	0.743 (~)	1 (~)	1 (~)
GWO$_{Crossover\_fbd}$ vs GWO$_{Mutation}$	0.273	0.785 (~)	1 (~)	1 (~)
GWO$_{Mutation}$ vs GWO$_{Crossover\_rbd}$	0.055	0.956 (~)	1 (~)	1 (~)
GWO$_{rbd}$ vs GWO$_{fbd}$	0	1 (~)	1 (~)	1 (~)

4.6 Real-world case studies

In this section, the proposed algorithms are used to solve real-life cases of which the data is taken from the related publications [36, 37].

The first application is concerned with the real-life implementation of the UFLP. Turkoglu, Genevois and Cedolin [36] have recently modeled an Automated Teller Machine (ATM) deployment problem as UFLP. ATM deployment problem is recognized as one of the significant strategic problems for the banking industry. The authors have considered Beşiktaş, which is a municipality in İstanbul, with a population of 186,570 and 23 districts. The goal of the study is to deploy the ATMs to potential candidate locations in order to serve the residents of such districts with the minimum cost. In addition to ATM deployment cost, the distance between the centers of the districts and the ATM locations should be minimized. The authors have used a great circle distance metric by using latitude and longitude data for ATMs locations and centers of the districts. Demand points are considered as customers, and ATM deployment locations are regarded as facilities in the UFLP. There are 22 ATM deployment locations and 23 customers in the mentioned study. All required latitude and longitude data are retrieved from the case study in [36]. The proposed algorithms are run for 30 replications, and the maximum number of iterations is set to 100. The obtained results are given in Table 16.

Table 16. Computational results for the ATM deployment problem

	Standard GWO	GWO$_{fbd}$	GWO$_{rbd}$
best	100,734.17	100,734.17	100,734.17
mean	100,931.03	100,802.32	100,734.17
worst	103,126.41	101,385.01	100,734.17
std.	398.62	219.06	0
hit	23	28	**30**

According to Table 16, the GWO$_{rbd}$ produces the best results in terms of hit values and is slightly better than GWO$_{fbd}$. Both of the proposed algorithms achieve the best results within a few iterations. On the other hand, the standard GWO algorithm with the sigmoidal transfer function exhibits high variance, which impacts the efficiency of the algorithm. Hence, the standard GWO algorithm could not return the best results in 7 times within the 30 replications. It is evident that the proposed algorithms improve the standard GWO and achieve better results in a real-life UFLP problem.

Another real-life case study is dedicated to the implementation of a 0-1 knapsack problem (KP). Ntow [37] has recently solved the problem of advertisement selection as 0-1 KP. The purpose of this problem is to select a subset of advertisements from a range of advertising items so that the advertisement reaches the public in the most efficient way. In this regard, Ntow [37] classifies spot ads as "Off-Peak Time," "Peak Time," "Premium Time," and "Premium" with 21, 18, 23, and 16 items, respectively. The knapsack capacity is assumed to be 360 secs for all cases. All other related data and parameters can be found in [37]. Eventually, the proposed algorithms are tested again for 30 replications of 100 iterations, and the results are given in Table 17.

Table 17. Computational results for the advertisement selection problem

cases	*perf.*	Standard GWO	GWO_{fbd}	GWO_{rbd}
off-peak time	*best*	4090	4090	4090
	mean	4090	4090	4090
	hit	30	30	30
peak time	*best*	5760	5760	5760
	mean	5760	5760	5760
	hit	30	30	30
prime time	*best*	9045	9045	9045
	mean	9045	9045	9045
	hit	30	30	30
premium time	*best*	10750	10750	10750
	mean	10750	10750	10750
	hit	30	30	30

It is evident from the findings of Table 17 that all algorithms perform equally well. This real-life-case study is a small-scaled problem so that the results are in line with the reported results of the small-scaled cases given earlier in Tables 4-6. The obtained results frankly show that the proposed methods successfully solve the real-life advertisement selection problem.

In short, both GWO_{fbd} and GWO_{rbd} are capable of finding promising results in real-life instances. One can also infer that the proposed modifications are effective and improve the standard GWO for binary problems. The proposed algorithms can be used to tackle real-life cases with high performance.

5. Concluding remarks

The present study proposes a simple and effective GWO algorithm to solve binary optimization problems. The proposed algorithm uses a direct representation of the binary solution vector without any need to transfer function. Therefore, the problem of spatial disconnect is avoided, and hence a more practical search mechanism is developed. Moreover, evolutionary and adaptive inheritance mechanisms are integrated into the procedural steps of the GWO algorithm. To this end, MP-Xover operation is carried out in order to effectively explore the binary search space. Next, to mimic the leadership hierarchy of the GWO algorithm, two different dominance strategies, which are referred to as fitness-based and rank-based dominance, are developed in the present study. The mentioned MP-Xover is followed by an adaptive mutation operation with an exponentially decreasing step-size to avoid premature convergence and to increase diversity within the wolf pack. Furthermore, more intensified search around the promising regions is ensured towards the end of iterations.

The performance of the proposed algorithm is tested on some of the well-known binary optimization problems, including SUKP and UFLP. A comprehensive experimental study, which is further validated by appropriate nonparametric statistical tests, is conducted. According to the obtained results, the proposed algorithm has achieved superior results in 8 out of 30 benchmarks in comparison with state-of-the-art swarm intelligence-based metaheuristics in the SUKP. Secondarily, the proposed algorithm successfully obtains the best solutions in the UFLP instances. Moreover, statistically verified significant improvements have been achieved over some existing algorithms reported in the related literature. Finally, developed algorithms are tested in two different real-life case studies. In future studies, the developed algorithms can be implemented in a wide array of binary optimization problems. Also, self-adaptive mechanisms can be incorporated for the parameters of the algorithm. Furthermore, hyper-heuristics and learning can be integrated in order to enhance the search capability of the algorithm. These considerations are scheduled as future research.

References

[1] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, IEEE, 1995, pp. 39-43.(https://doi.org/10.1109/MHS.1995.494215)
[2] M. Dorigo, G. Di Caro, Ant colony optimization: a new meta-heuristic, In: Proceedings of the 1999 congress on evolutionary computation, IEEE, 1999, pp. 1470-1477. (https://doi.org/10.1109/CEC.1999.782657)
[3] X.-S. Yang, Firefly algorithms for multimodal optimization, In: O. Watanabe, T. Zeugmann (Eds.), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2009, pp. 169-178. (https://doi.org/10.1007/978-3-642-04944-6_14)
[4] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, J. Global Optim. 39 (2007) 459-471. (https://doi.org/10.1007/s10898-007-9149-x)
[5] S. Mirjalili, A. Lewis, The Whale Optimization Algorithm, Adv. Eng. Softw. 95 (2016) 51-67. (https://doi.org/10.1016/j.advengsoft.2016.01.008)
[6] X.-S. Yang, Flower pollination algorithm for global optimization, In: J. Durand-Lose, N. Jonoska (Eds.), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 240-249. (https://doi.org/10.1007/978-3-642-32894-7_27)
[7] X.-S. Yang, S. Deb, Cuckoo search: recent advances and applications, Neural. Comput. Appl. 24 (2014) 169-174. (https://doi.org/10.1007/s00521-013-1367-1)
[8] S. Mirjalili, S.M. Mirjalili, A. Lewis, Grey Wolf Optimizer, Adv. Eng. Softw. 69 (2014) 46-61. (https://doi.org/10.1016/j.advengsoft.2013.12.007)
[9] S. Luo, L. Zhang, Y. Fan, Energy-efficient scheduling for multi-objective flexible job shops with variable processing speeds by grey wolf optimization, J. Clean. Prod. 234 (2019) 1365-1384. (https://doi.org/10.1016/j.jclepro.2019.06.151)
[10] J. Oliveira, P.M. Oliveira, J. Boaventura-Cunha, T. Pinho, Chaos-based grey wolf optimizer for higher order sliding mode position control of a robotic manipulator, Nonlinear Dynamics, 90 (2017) 1353-1362. (https://doi.org/10.1007/s11071-017-3731-7)

[11] N. Singh, S.B. Singh, A Modified mean Gray Wolf Optimization approach for benchmark and biomedical problems, Evol. Bioinform. 13 (2017) 1-28. (https://doi.org/10.1177/1176934317729413)
[12] F.B. Ozsoydan, Effects of dominant wolves in grey wolf optimization algorithm, Appl. Soft. Comput. 83 (2019) 105658. (https://doi.org/10.1016/j.asoc.2019.105658)
[13] T. Jiang, C. Zhang, Application of Grey Wolf Optimization for solving combinatorial problems: job shop and flexible job shop scheduling cases, IEEE Access, 6 (2018) 26231-26240. (https://doi.org/10.1109/ACCESS.2018.2833552)
[14] M. Wang, H. Chen, H. Li, Z. Cai, X. Zhao, C. Tong, J. Li, X. Xu, Grey wolf optimization evolving kernel extreme learning machine: Application to bankruptcy prediction, Eng. Appl. Artif. Intel. 63 (2017) 54-68. (https://doi.org/10.1016/j.engappai.2017.05.003)
[15] S. Mirjalili, S. Saremi, S.M. Mirjalili, L.S. Coelho, Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization, Expert. Syst. Appl. 47 (2016) 106-119. (https://doi.org/10.1016/j.eswa.2015.10.039)
[16] S. Mirjalili, How effective is the Grey Wolf optimizer in training multi-layer perceptrons, Appl. Intel. 43 (2015) 150-161. (https://doi.org/10.1007/s10489-014-0645-7)
[17] L.M.Q. Abualigah, Feature Selection and Enhanced Krill Herd Algorithm for Text Document Clustering, Springer International Publishing, Switzerland, 2019.
[18] L.M. Abualigah, A.T. Khader, E.S. Hanandeh, Hybrid clustering analysis using improved krill herd algorithm, Appl. Intel. 48 (2018) 4047-4071. (https://doi.org/10.1007/s10489-018-1190-6)
[19] L.M. Abualigah, A.T. Khader, E.S. Hanandeh, A combination of objective functions and hybrid Krill Herd Algorithm for text document clustering analysis, Eng. Appl. Artif. Intel. 73 (2018) 111-125. (https://doi.org/10.1016/j.engappai.2018.05.003)
[20] L.M.Q. Abualigah, E.S. Hanandeh, Applying genetic algorithms to information retrieval using vector space mode, Int. J. Comput. Sci. Eng. Appl. 5 (2015) 19-28. (https://doi.org/10.5121/ijcsea.2015.5102)
[21] L.M. Abualigah, A.T. Khader, Unsupervised text feature selection technique based on hybrid particle swarm optimization algorithm with genetic operators for the text clustering, The Journal of Supercomputing, 73 (2017) 4773-4795. (https://doi.org/10.1007/s11227-017-2046-2)
[22] L.M. Abualigah, A.T. Khader, E.S. Hanandeh, A new feature selection method to improve the document clustering using particle swarm optimization algorithm, J. Comput. Sci-Neth. 25 (2018) 456-466. (https://doi.org/10.1016/j.jocs.2017.07.018)
[23] J. Liu, Y. Mei, X. Li, An analysis of the inertia weight parameter for binary particle swarm optimization, IEEE T. Evolut. Comput. 20 (2016) 666-681. (https://doi.org /10.1109/TEVC.2015.2503422)
[24] A. Jaszkiewicz, On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - a comparative experiment, IEEE T. Evolut. Comput. 6 (2002) 402-412. (https://doi.org /10.1109/TEVC.2002.802873)
[25] L. Yu, S. Wang, F. Wen, K.K. Lai, Genetic algorithm-based multi-criteria project portfolio selection, Ann. Oper. Res. 197 (2012) 71-86. (https://doi.org/10.1007/s10479-010-0819-6)
[26] S. Wang, J. Watada, A hybrid modified PSO approach to VaR-based facility location problems with variable capacity in fuzzy random uncertainty, Inform. Sciences. 192 (2012) 3-18. (https://doi.org/10.1016/j.ins.2010.02.014)
[27] J. Too, A.R. Abdullah, N.M. Saad, N.M. Ali, W. Tee, A new competitive binary Grey Wolf Optimizer to solve the feature selection problem in EMG signals classification, Computers. 7 (2018) 58. (https://doi.org/10.3390/computers7040058)

[28] V.B. Pamshetti, S. Singh, S.P. Singh, Reduction of energy demand via conservation voltage reduction considering network reconfiguration and soft open point, Int. T. Electr. Energy. 0 (2019) e12147. (https://doi.org/10.1002/2050-7038.12147)
[29] S.T. Veena, S. Arivazhagan, W.S.L. Jebarani, Improved detection of steganographic algorithms in spatial LSB stego images using hybrid GRASP-BGWO optimisation, In: J. Hemanth, V. Balas (Eds.), Nature Inspired Optimization Techniques for Image Processing Applications. Intelligent Systems Reference Library, Springer, Cham, 2019, pp. 89-112. (https://doi.org/10.1007/978-3-319-96002-9_4)
[30] S. Velliangiri, A hybrid BGWO with KPCA for intrusion detection, J. Exp. Theor. Artif In. (2019) 1-16. (https://doi.org/10.1080/0952813X.2019.1647558)
[31] Q. Al-Tashi, S.J.A. Kadir, H.M. Rais, S. Mirjalili, H. Alhussian, Binary optimization using hybrid Grey Wolf Optimization for feature selection, IEEE Access, 7 (2019) 39496-39508. (https://doi.org/10.1109/ACCESS.2019.2906757)
[32] B.J. Leonard, A.P. Engelbrecht, C.W. Cleghorn, Critical considerations on angle modulated particle swarm optimisers, Swarm. Intel. 9 (2015) 291-314. (https://doi.org/10.1007/s11721-015-0114-x)
[33] O. Goldschmidt, D. Nehme, G. Yu, Note: On the set-union knapsack problem, Nav. Res. Log. 41 (1994) 833-842. (https://doi.org/10.1002/1520-6750(199410)41:6<833::AID-NAV3220410611>3.0.CO;2-Q)
[34] M.L. Cornuejols, N.G. L., L.A. Wolsey, The Uncapacitated Facility Location Problem, In: R.L. Francis, P. Mirchandani (Eds.) Discrete location theory, Wiley Interscience, New York, 1990.
[35] Y. He, H. Xie, T.-L. Wong, X. Wang, A novel binary artificial bee colony algorithm for the set-union knapsack problem, Future. Gener. Comput. Sy. 78 (2018) 77-86. (https://doi.org/10.1016/j.future.2017.05.044)
[36] D.C. Turkoglu, M.E. Genevois, M. Cedolin, Facility location problems-a case study for ATM site selection, In: Proceeding of the 12th Multidisciplinary Academic Conference (MAC), Czech Republic, Prague., 2018.
[37] A.S. Ntow, Knapsack problem a case study of metro t.v., a television station in Ghana, In: Industrial mathematics, Kwame Nkrumah University of Science and Technology, Kumasi, Ghana, 2012.
[38] M. Yu, X. Li, J. Liang, A dynamic surrogate-assisted evolutionary algorithm framework for expensive structural optimization, Struct. Multidiscip. O. (*in press*). (https://doi.org/10.1007/s00158-019-02391-8)
[39] A. Arulselvan, A note on the set union knapsack problem, Discrete. Appl. Math. 169 (2014) 214-218. (https://doi.org/10.1016/j.dam.2013.12.015)
[40] F.B. Ozsoydan, A. Baykasoglu, A swarm intelligence-based algorithm for the set-union knapsack problem, Future. Gener. Comput. Sy. 93 (2019) 560-569. (https://doi.org/10.1016/j.future.2018.08.002)
[41] C. Wu, Y. He, Solving the set-union knapsack problem by a novel hybrid Jaya algorithm, Soft Comput. (*in press*). (https://doi.org/10.1007/s00500-019-04021-3)
[42] Y. Feng, H. An, X. Gao, The importance of transfer function in solving set-union knapsack problem based on discrete moth search algorithm, Mathematics, 7 (2018) 17. (https://doi.org/10.3390/math7010017)
[43] A. Baykasoğlu, F.B. Ozsoydan, M.E. Senol, Weighted superposition attraction algorithm for binary optimization problems, Oper. Res. (*in press*). (https://doi.org/10.1007/s12351-018-0427-9)
[44] Y. He, X. Wang, Group theory-based optimization algorithm for solving knapsack problems, Knowl-Based. Syst. (*in press*). (https://doi.org/10.1016/j.knosys.2018.07.045)

[45] J.H. Jaramillo, J. Bhadury, R. Batta, On the use of genetic algorithms to solve location problems, Comput. Oper. Res. 29 (2002) 761-779. (https://doi.org/10.1016/S0305-0548(01)00021-1)
[46] K.S. Al-Sultan, M.A. Al-Fawzan, A tabu search approach to the uncapacitated facility location problem, Ann. Oper. Res. 86 (1999) 91-103. (https://doi.org/10.1023/A:1018956213524)
[47] D. Wang, C.-H. Wu, A. Ip, D. Wang, Y. Yan, Parallel multi-population Particle Swarm Optimization algorithm for the uncapacitated facility location problem using OpenMP, In: IEEE Congress on Evolutionary Computation, 2008, pp. 1214-1218. (https://doi.org/10.1109/CEC.2008.4630951)
[48] A.R. Güner, M. Şevkli, A discrete particle awarm optimization algorithm for uncapacitated facility location problem, J. Artif. Evolut. Appl. (2008) 1-9. (https://doi.org/10.1155/2008/861512)
[49] M. Şevkli, A.R. Güner, A continuous particle swarm optimization algorithm for uncapacitated facility location problem, In: M. Dorigo, L.M. Gambardella, M. Birattari, A. Martinoli, R. Poli, T. Stützle (Eds.), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2006, pp. 316-323. (https://doi.org/10.1007/11839088_28)
[50] K. Tsuya, M. Takaya, A. Yamamura, Application of the firefly algorithm to the uncapacitated facility location problem, J. Intell. Fuzzy Syst. 32 (2017) 3201-3208. (https://doi.org/10.3233/JIFS-169263)
[51] M.S. Kıran, The continuous artificial bee colony algorithm for binary optimization, Appl. Soft. Comput. 33 (2015) 15-23. (https://doi.org/10.1016/j.asoc.2015.04.007)
[52] F.B. Ozsoydan, Artificial search agents with cognitive intelligence for binary optimization problems, Comput. Ind. Eng. 136 (2019) 18-30. (https://doi.org/10.1016/j.cie.2019.07.007)
[53] T.S. Hale, C.R. Moberg, Location science research: a review, Ann. Oper. Res. 123 (2003) 21-35. (https://doi.org/10.1023/A:1026110926707)
[54] G. Şahin, H. Süral, A review of hierarchical facility location models, Comput. Oper. Res. 34 (2007) 2310-2331. (https://doi.org/10.1016/j.cor.2005.09.005)
[55] D. Simon, Evolutionary Optimization Algorithms, John Wiley & Sons, 2013.
[56] H.-P. Schwefel, Evolution and Otimum Seeking: The Sixth Generation, John Wiley & Sons, Inc., 1993.
[57] A.E. Eiben, C.A. Schippers, Multi-parent's niche: N-ary crossovers on NK-landscapes, In: H.M. Voigt, W. Ebeling, I. Rechenberg, H.P. Schwefel (Eds.), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1996, pp. 319-328. (https://doi.org/10.1007/3-540-61723-X_996)
[58] T. Bäck, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford university press, 1996.
[59] A.E. Eiben, Multiparent recombination in evolutionary computing, in: A. Ghosh, S. Tsutsui (Eds.), Advances in Evolutionary Computing, Natural Computing Series, Springer, Berlin, Heidelberg, 2003, pp. 175-192. (https://doi.org/10.1007/978-3-642-18965-4_6)
[60] A. Baykasoğlu, Design optimization with chaos embedded great deluge algorithm, Appl. Soft. Comput. 12 (2012) 1055-1067. (https://doi.org/10.1016/j.asoc.2011.11.018)
[61] A. Baykasoğlu, F.B. Ozsoydan, Adaptive firefly algorithm with chaos for mechanical design optimization problems, Appl. Soft. Comput. 36 (2015) 152-164. (https://doi.org/10.1016/j.asoc.2015.06.056)
[62] M. Sevkli, A.R. Guner, A continuous particle swarm optimization algorithm for uncapacitated facility location problem, In: M. Dorigo, L.M. Gambardella, M. Birattari, A. Martinoli, R. Poli, T. Stützle (Eds.), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2006, pp. 316-323. (https://doi.org/10.1007/11839088_28)

[63] M.S. Kiran, The continuous artificial bee colony algorithm for binary optimization, Appl. Soft. Comput. 33 (2015) 15-23. (https://doi.org/10.1016/j.asoc.2015.04.007)
[64] A. Hamzadayı, Ş. Topaloğlu, S.Y. Köse, Nested simulated annealing approach to periodic routing problem of a retail distribution system, Comput. Oper. Res. 40 (2013) 2893-2905. (https://doi.org/10.1016/j.cor.2013.06.004)
[65] H. Mosadegh, M. Zandieh, S.M.T.F. Ghomi, Simultaneous solving of balancing and sequencing problems with station-dependent assembly times for mixed-model assembly lines, Appl. Soft. Comput. 12 (2012) 1359-1370. (https://doi.org/10.1016/j.asoc.2011.11.027)
[66] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm. Evol. Comput. 1 (2011) 3-18. (https://doi.org/10.1016/j.swevo.2011.02.002)

CRediT author statement

İlker Gölcük: Conceptualization, Methodology, Data curation, Formal analysis, Software, Writing - Original Draft, Writing - Review & Editing, Visualization.

Fehmi Burcin Ozsoydan: Conceptualization, Methodology, Data curation, Writing - Review & Editing, Validation, Investigation, Resources, Supervision

Declaration of interests

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

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Highlights

- A new binary grey wolf optimization algorithm is developed
- Fitness-based and rank-based dominance strategies are developed
- Multi-parent crossover and adaptive mutation are utilized
- Set-Union Knapsack Problem and Uncapacitated Facility Location Problem are solved
- Effectiveness of the algorithm is statistically verified