On the binarization of Grey Wolf optimizer: a novel binary optimizer algorithm

Mehdy Roayaei

Accepted: 14 September 2021 / Published online: 30 September 2021
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2021

Abstract
Grey Wolf Optimizer (GWO) is a nature-inspired swarm intelligence algorithm that mimics the hunting behavior of grey wolves. GWO, in its basic form, is a real coded algorithm that needs modifications to deal with binary optimization problems. In this paper, previous work on the binarization of GWO are reviewed, and are classified with respect to their encoding scheme, updating strategy, and transfer function. Then, we propose a novel binary GWO algorithm (named SetGWO), which is based on set encoding and uses set operations in its updating strategy. The proposed algorithm uses a completely different encoding scheme that eliminates the need for the transfer function and boundary checking, and also uses lower-dimensional agents; therefore, decreases the running time. Also, by using an exclusive exploration set for each agent, defining a different distance measure and an encircling strategy in discrete spaces, the quality of solutions has been improved. Experimental results on different real-world combinatorial optimization problems and datasets show that SetGWO outperforms other existing binary GWO algorithms in terms of quality of solutions, running time, and scalability.

Keywords Grey Wolf Optimizer · Binary Combinatorial Optimization · Swarm Intelligence · Metaheuristics

1 Introduction
Combinatorial optimization is a category of optimization that consists of finding an optimal object from a finite set of objects. It operates on the domain of those optimization problems in which the set of feasible solutions is discrete. In this paper, we focus on binary optimization problems, where the goal is to find a subset of size \( k \) (for a given integer \( k \)) of a given set of elements to maximize (or minimize) an objective function. Dominating set (Fomin et al. 2004), minimum spanning tree (Katagiri et al. 2012), feature selection (Abuali-gah 2019), and 0/1 Knapsack (Abdel-Basset et al. 2019) are a few to mention.

Metaheuristics are general algorithmic frameworks, often nature-inspired swarm intelligence, designed to solve complex optimization problems (Abualighah et al. 2021; Abualighah and Diabat 2021; Abualighah et al. 2021). Swarm intelligence (SI) is one of the optimization approaches which are used to solve complex optimization problems. SI focuses on the collective behaviors of the individuals, especially nature-inspired agents, that result from the local interactions of the individuals with each other. Agents obey simple rules, and no centralized mechanism exists to predict the behavior of individual agents. Although there is no centralized structure for individual agents’ behavior, local, and to a certain degree random interactions between the agents provide an intelligent global behavior which is then unknown to individual agents. Some of popular SI algorithms include Particle Swarm Optimization (Bello et al. 2007), Artificial Bee Colony (Clodomir 2019), Firefly Algorithm (Bhattacharjee and Sarmah 2015), and Cuckoo Search (Kaya 2018). Swarm intelligence has many applications such as in healthcare (Zemmal et al. 2020; Sharma et al. 2020), query optimization (Sharma et al. 2019), cyber-physical systems (Schranz et al. 2021), Internet of Things (El-Shafeiy et al. 2021), task scheduling (Boveiri et al. 2019), and sentiment analysis (Kumar et al. 2016).

Grey wolf optimizer (GWO) (Mirjalili et al. 2014) is one of the nature-inspired swarm intelligence algorithms that has been widely tailored for a wide variety of optimization problems due to its impressive characteristics over other metaheuristics. It has very few parameters, and no derivation information is required in the initial search. Also, it is simple, easy to use, flexible, scalable, and has a special capability to
strike the right balance between exploration and exploitation during the search, which leads to favorable convergence.

GWO, in its basic form, is a real coded algorithm and therefore needs modifications to deal with binary optimization problems. There have been many efforts in the literature on the binarization of GWO in recent years. However, there are some limitations in previous works that should be handled. The main drawback is that most of the previous binary versions of GWO preserve the exploration and exploitation strategies of the basic GWO, and use some functions to binarize the resulting values at the end of each iteration. The experimental results show that it leads to premature convergence. It seems that these strategies must be adapted to fit the characteristics of discrete spaces. Also, two factors in previous works increase the running time. First, in all previous algorithms, each agent is represented by a vector of length the size of the problem instead of the size of the solution. Note that all operations in exploration and exploitation strategies must be applied to all dimensions of all agents in all iterations. Second, most of these algorithms use a transfer function to convert real values to binary values. Again, the results show that these two factors significantly increase the running times of algorithms.

In this paper, binary variations of GWO are reviewed, focusing on their encoding scheme, updating strategy, and transfer function. Then, we propose a novel binary GWO (named SetGWO), which is based on set encoding and uses set operations in its updating strategy. Experimental results on different real-world combinatorial optimization problems (Influence Maximization, Vertex Cover, and 0/1 Knapsack) and datasets (18 datasets) show that SetGWO outperforms other existing binary GWO algorithms in terms of quality of solutions, running time, and scalability. The novelty of the proposed algorithm can be summarized as follows:

- The proposed algorithm uses a completely different encoding scheme (set encoding). It uses lower-dimensional agents and eliminates the need for the transfer function and boundary checking, which as a result decreases the running time.
- By using an exclusive exploration set for each agent, defining a new distance measure to better determine the distance of two agents in a discrete space, and modifying the encircling strategy to do more meaningful exploitation in discrete spaces, the quality of solutions has been improved.

In the remainder of the paper, the set of all input elements is denoted by $S$, where $|S| = n$. The parameter $k$ denotes the maximum possible size of a selected subset. Also, $k$-set denotes a set of size $k$. Furthermore, rand($S,k$) denotes a random $k$-subset of $S$.

The remainder of the paper is organized as follows. In Sect. 2, the basic GWO is introduced, and related work on the binarization of the basic GWO is reviewed. The proposed binary version of GWO is described in Sect. 3. The experimental results are discussed in Sect. 4. Finally, conclusions are stated in Sect. 5.

2 Related work

GWO is a metaheuristic that mimics the hierarchical leadership and hunting strategy of grey wolves in nature (Mirjalili et al. 2014). Four types of grey wolves are employed for simulating the leadership hierarchy: alpha($\alpha$), beta($\beta$), delta($\delta$), and omega($\omega$). The best three grey wolves are considered alpha, beta, and delta, and the remaining grey wolves are termed omega. GWO simulates the major steps of grey wolves hunting, searching, encircling, and attacking. The hunting is guided by $\alpha$, $\beta$, and $\delta$. The $\omega$ wolves follow these three wolves.

GWO works as follows: the initial population (a pack of wolves) is generated. The position of each wolf, which is represented as a vector, is a candidate solution. That is, each wolf is represented as a point in a multi-dimensional space. The best three wolves (leaders) are selected as $\alpha$, $\beta$, and $\delta$, respectively. Other wolves ($\omega$ wolves) update their positions according to the position of the leaders. At the end of each iteration, the three best wolves are selected as new leaders, and the next iteration starts. The algorithm goes on until a condition of termination is reached.

Equations (2.1) and (2.2) mathematically model encircling behavior of GWO:

$$D = |C.X_p(t) - X(t)| \quad (2.1)$$
$$X(t + 1) = X_p(t) - A.D \quad (2.2)$$

where $t$ indicates the current iteration, $X_p$ is the position vector of the prey, $X$ indicates the position vector of an omega wolf, and $A$ and $C$ are coefficient vectors, which are calculated as follows:

$$A = 2a.r_1 - a \quad (2.3)$$
$$C = 2.r_2 \quad (2.4)$$

where $r_1$ and $r_2$ are two independent random vectors in $[0, 1]$, and components of $a$, which is the encircling coefficient that is used to balance the tradeoff between exploration and exploitation, are linearly decreased from $2$ to $0$ over the course of iterations. Component $C$ provides random weights for prey in order to stochastically emphasize ($C > 1$) or deemphasize ($C < 1$) the effect of prey in defining the distance in Eq. (2.1).
Since the position of the prey (optimal solution) is not known in optimization problems, the three leaders guide the omega wolves to move toward the optimal solution. Thus, the position of an omega wolf is calculated as in Eq. (2.5):

\[
X(t + 1) = \frac{X_1(t) + X_2(t) + X_3(t)}{3},
\]

where \(X_1, X_2,\) and \(X_3\) are calculated using Eqs. (2.1) and (2.2), considering that \(X_p\) is replaced with \(X_\alpha, X_\beta,\) and \(X_\delta,\) respectively. Figure 1 depicts the GWO updating strategy.

Because of its advantages, GWO has been successfully adapted for a wide range of optimization problems. Flow Shop Scheduling (Komaki and Kayvanfar 2015), Vehicle Path Planning (Zhang et al. 2016), Feature Selection (Al-Tashi et al. 2020), Multi-dimensional Knapsack (Luo and Zhao 2019), Numeric Optimization (Long et al. 2020), Traveling Salesman Problem (Sopto et al. 2018), Signal Processing (Rao and Malathi 2019), and Text Classification (Chantar et al. 2020) are just a few to mention.

The basic GWO is a real coded algorithm and was originally designed to tackle continuous optimization problems. Therefore, it needs some modifications to deal with binary optimization problems, where binary space is used and solutions are limited to 0 and 1 values. Hence, there have been many efforts in recent years to adapt the basic GWO for binary optimization problems.

Each binarization approach for GWO must consider three critical components. The first component is encoding scheme, which must provide a scheme for encoding solutions of a binary optimization problem in the form of GWO components. The second component is updating strategy, which must provide a strategy to update the positions of omega wolves. The third component is transfer function, which shows how to convert a real number to 0 or 1. In the following, related works on the binarization of GWO concerning these components will be reviewed.

One of the most cited works in this area is (Emary et al. 2016), in which, Emary et al. proposed two binary versions for realizing the optimal set of features in the feature selection problem. They used the binary encoding scheme in both versions. In the first approach, individual steps toward the leaders are binarized, and then stochastic crossover is performed among the three basic moves to find the updated binary grey wolf position. In the second approach, a sigmoid function is used to squash the continuous updated position, then stochastically threshold these values to find the updated binary grey wolf position. In the following years, their work was adapted by many others such as (Liu et al. 2020) and (Devanathan et al. 2019).

Manikandan et al. suggested new binary modifications of GWO (Manikandan et al. 2016). In their approach, where binary encoding was used, GWO is modified by binarizing only the initial three optimal solutions and updating the wolf position using stochastic crossover. Modifications were also carried out using sigmoid functions to compress the continuous updated positions.

Sahoo et al. proposed a binary GWO algorithm for the cervix lesion classification problem (Sahoo and Chandra 2017). They used binary encoding. As a transfer function, two steps were proposed to obtain binary representation for positions of grey wolves. First, \(tanh()\) function scales real coded position values to the range of \([-1, 1]\). Then, each scaled value is compared with a randomly generated threshold \(T \in \text{rand}(0, 1)\). If it exceeds the threshold value, the bit is set to 1; otherwise, it is set to 0.

Al-Tashi et al. proposed a hybrid binary GWO for the feature selection problem that benefits from the strengths of both GWO and PSO (Al-Tashi et al. 2019). They used binary encoding scheme, a combination of basic GWO and PSO as the updating strategy, and a sigmoid function as the transfer function.

Chantar et al. proposed an approach to convert the continuous GWO to binary version for enhancing feature selection in the text classification problem (Chantar et al. 2020). They used binary encoding scheme. They used a sigmoid function, as a transfer function, to binarize the movement vector of a grey wolf. For the updating strategy, they used an elite-based crossover operator to combine the three leaders instead of applying the conventional average operator.

Luo et al., to tackle the multi-dimensional knapsack problem, proposed a binary grey wolf optimizer, which integrates some important features including an initial elite population generator, a pseudo-utility-based quick repair operator, and a new evolutionary mechanism with a differentiated position updating strategy (Luo and Zhao 2019). They used...
binary representation for encoding solutions. For converting continuous values to binary values, they experimentally evaluated six different transfer functions and concluded that the absolute function of hyperbolic tangent performs better than others.

Hu et al. used binary encoding scheme to encode solutions (Hu et al. 2020). Also, they proposed five transfer functions for mapping continuous values to binary values, one sigmoid function and four different V-shape functions.

Zareie et al. proposed probabilistic encoding scheme, in which each solution (wolf) \( X_j \) is shown as a vector of \( n \) elements, where the value of \( j \)-th element indicates the chance of element \( j \) to be selected in the solution (Zareie et al. 2020). Thus, the corresponding solution contains \( k \) elements with the highest values in \( X_j \). They used the same updating strategy as in the basic GWO.

El-Kenawy et al. proposed a binary GWO based on Stochastic Fractal Search (SFS) to balance exploration and exploitation (El-Kenawy et al. 2020). They developed a modified GWO by applying an exponential form for the number of iterations of the original GWO to improve exploitation, and crossover/mutation operations to enhance exploitation capability. Also, they applied the diffusion procedure of SFS for the best solution of the modified GWO by using the Gaussian distribution method for random walk in a growth process. The continuous values of their algorithm were then converted into binary values using a sigmoid function.

In Rebello and de Oliveira (2020), Rebello et al. proposed the use of a sigmoid transfer function to convert the optimization variables into binary values. They also incorporated a simple modification at the local search component of GWO to best suit its application to escape local minima.

There have been many modifications to the basic GWO in previous work. However, considering the encoding scheme, updating strategy, and transfer function, previous work can be summarized as in Table 1.

As can be seen, the previous encoding schemes can be divided into two categories:

- **Binary**, where each element (dimension) is represented by 1 or 0, which indicates whether the element is selected as a part of the corresponding solution or not.
- **Probabilistic**, where each element is represented as a real number \( \epsilon [0,1] \), which indicates the probability of selecting the element in the corresponding solution.

Also, the previous updating strategies can be divided into three categories:

- **Basic GWO**, where the same updating strategy as in the basic GWO is used to update the position of an omega wolf.
- **Arithmetic**, where the basic GWO updating strategy has been modified, but the new position of an omega wolf is still calculated using simple arithmetic operations on the positions of the leaders.
- **Crossover**, where crossover is performed between solutions and three leaders to find the updated binary position of an omega wolf.

Also, different transfer functions such as V-shape, sigmoid, threshold, and tanh() have been used in previous work to convert real numbers to binary values.

Considering the basic GWO and previous work on the binarization of GWO, their disadvantages can be summarized as follows:

- In all previous algorithms, each wolf is represented by a vector of length \( n \) (size of the problem) instead of \( k \) (size of the solution), which increases the running time. Thus, it seems that the encoding scheme can be improved.
- Algorithms that use transfer functions to convert real values to integer values are much slower than others. A transfer function must be called for all dimensions of all wolves in all iterations. Thus, it seems that the encoding scheme can be improved.
- The position of an omega wolf is updated according to the average position of the three leaders. But, in a discrete space, the average position does not necessarily lead to a discrete value or even a feasible solution. Thus, it seems that the encircling strategy must be adapted.
- In exploitation (exploration), the distance of an omega wolf from the average position of the leaders is decreased (increased). Because a real value cannot determine whether its corresponding element is in the solution or not, the basic exploitation (exploration) strategy does not seem applicable in discrete spaces, where each dimension of a wolf position is 0 or 1. Thus, it seems that the distance measure and exploration and exploitation strategies must be adapted.

### 3 Proposed algorithm

In this section, we propose SetGWO, a novel binary optimizer based on the basic GWO. In this algorithm, set encoding, where each wolf is represented as a set (a \( k \)-subset of \( S \)) has been used. For updating strategy, only simple set operators (union, intersection, and difference) are used. Because of using such an encoding scheme and updating strategy, there is no need to use a transfer function. The same parameters \((C, A, r_1, r_2)\) as in the basic GWO are used. The flowchart of the proposed algorithm is shown in Fig. 2.
### Table 1  Summarization of previous work on binary GWO

| Author        | Encoding Scheme | Updating Strategy | Transfer function | Ref.                                      |
|---------------|-----------------|-------------------|-------------------|-------------------------------------------|
| Emary et al.  | Binary          | Crossover         | None              | (Emary et al. 2016)                       |
| Emary et al.  | Binary          | Basic GWO         | Sigmoid           | (Emary et al. 2016)                       |
| Manikandan et al. | Binary      | Crossover         | Sigmoid           | (Manikandan et al. 2016)                  |
| Sahoo et al.  | Binary          | Basic GWO         | Tanh()            | (Sahoo and Chandra 2017)                  |
| Al-Tashi et al.| Binary          | Basic GWO         | Tanh()            | (Al-Tashi et al. 2019)                    |
| Chantar et al.| Binary          | Basic GWO         | Sigmoid           | (Chantar et al. 2020)                     |
| Lu et al.     | Binary          | Arithmetic        | Tanh()            | (Luo and Zhao 2019)                       |
| Tu et al.     | Binary          | Arithmetic        | Threshold         | (Tu et al. 2019)                          |
| Hu et al.     | Binary          | Arithmetic        | Sigmoid,V-shape   | (Hu et al. 2020)                          |
| Zareie et al. | Probabilistic   | Basic GWO         | None              | (Zareie et al. 2020)                      |
| El-kenawy et al. | Binary     | Crossover         | Sigmoid           | (El-kenawy et al. 2020)                   |
| Rebello et al.| Binary          | Arithmetic        | Sigmoid           | (Rebello and de Oliveira 2020)            |

Fig. 2  Flowchart of SetGWO

The main components of the proposed algorithm are described in detail in the following.

#### 3.1 Initialization and encoding

Each wolf is represented by a $k$-set, so the order of elements does not matter. In the initialization phase, each wolf is initialized by a random $k$-subset of $S$.

#### 3.2 Encircling prey

In the basic GWO, in which wolves are modeled as points in a continuous space, the position of an omega wolf is updated according to the average position of the three leaders. But, in a discrete space, the average position does not necessarily lead to a discrete value. Furthermore, the average value of leaders’ positions does not necessarily lead to a good or even a feasible solution. In SetGWO, omegas are updated according to one leader (the closest leader to the omega) as in Fig. 3.

#### 3.3 Distance measure

Since each omega is represented as a set, the order of its elements does not matter, and considering it as a point in a $k$-dimensional space is meaningless. Thus, the distance function has been redefined. The distance of omega $X_i$ to a leader is defined as the number of elements in $X_i$ which are not in the leader (that is their set difference) as in Eq. (3.1):
Encircling strategy in SetGWO

\[ D(X_i, \text{leader}) = X_i - \text{leader} \] (3.1)

3.4 Updating strategy

Since omegas are represented as sets, simple set operations (union, intersection, and difference) are used to update an omega. As in the basic GWO, if \(|A| < 1\), omega converges toward its closest leader, and if \(|A| > 1\), omega diverges from its closest leader to hopefully find a better solution.

**Exploitation**

In exploitation, to decrease the distance of an omega from a leader, some elements of \(X_i\) that are not in the leader are substituted with the same number of elements of the leader that are not in \(X_i\), as in Eq. (3.2).

\[ N = \text{leader} - X_i \]
\[ O = X_i - \text{leader} \]
\[ D = |O| \]
\[ \text{step} = (1 - |A|) \times D \]
\[ X_i = X_i - \text{rand}(O, \text{step}) \]
\[ X_i = X_i \cup \text{rand}(N, \text{step}) \] (3.2)

where \(N\) is the set of elements of the leader that are not in \(X_i\), \(O\) is the set of elements of \(X_i\) that are not in the leader, \(D\) is the distance of the omega from its leader, and \(\text{step}\) is the extent of which the distance \(D\) is decreased after convergence.

**Exploration**

In exploration, to increase the distance of an omega from a leader, the elements of \(X_i\) that are in the leader are substituted with the elements of \(S\) that are neither in \(X_i\) nor in the leader. Also, to improve exploration, we maintain an exclusive exploration set for each omega \(X_i\), denoted by \(S_i\), which contains the elements of \(S\) that are not tried by \(X_i\) yet. \(S_i\) is initialized with \(S\) at the beginning of the algorithm and refilled when its size is below a given threshold.

\[ N = S_i - (X_i \cup \text{leader}) \]
\[ O = \text{leader} \cap X_i \]
\[ D = |X_i - \text{leader}| \]
\[ \text{step} = (1 - |A|) \times D \]
\[ X_i = X_i - \text{rand}(O, \text{step}) \]
\[ S_{\text{step}} = \text{rand}(N, \text{step}) \]
\[ X_i = X_i \cup S_{\text{step}} \]
\[ S_i = S_i - S_{\text{step}} \] (3.3)

where \(N\) is the set of elements of \(S_i\) which are neither in \(X_i\) nor in the leader, \(O\) is the set of elements which are in both \(X_i\) and leader, \(D\) is the distance of the omega from the leader, \(\text{step}\) is the extent of which the distance \(D\) is increased after divergence, and \(S_{\text{step}}\) is the set of elements that are added to \(X_i\) as new elements.

After all, the pseudo-code of the proposed algorithm is provided in Algorithm SetGWO.

4 Result and discussion

In this section, SetGWO is benchmarked on three binary optimization problems and 18 datasets. To evaluate the proposed algorithm, three different binary versions of GWO are implemented, which obtained better results in the literature and use different schemes and strategies, with the following settings:

- **BCROSS** (Emary et al. 2016): binary encoding scheme, crossover updating strategy, no transfer function.
- **BGWO** (Emary et al. 2016): binary encoding scheme, basic GWO updating strategy, and a sigmoid function as transfer function.
- **BPROB** (Zareie et al. 2020): probabilistic encoding scheme, basic GWO updating strategy, no transfer function.

Although the main focus of this paper is on improving the binarization of GWO, five other non-GWO binary optimizer algorithms are implemented to compare with:

- **BGA** (Kabir et al. 2011): binary genetic algorithm.
- **BPSO** (Bello et al. 2007): binary particle swarm optimization.
Algorithm SetGWO

Input: |S|: set of all elements, k: size of a solution, max_it: number of iterations, pop_size: number of wolves, and t: threshold.

1: Initialize the population as random k-subsets of S
2: for 1 ≤ i ≤ pop_size do
3: S_i = S
4: end for
5: for 1 ≤ it ≤ max_it do
6: calculate the fitness of all wolves
7: X_α = the best wolf
8: X_β = the second best wolf
9: X_δ = the third best wolf
10: a = 2 - it × (max.it)
11: for 1 ≤ i ≤ pop_size do
12: if (|S_i| < t × k) then
13: S_i = S
14: end if
15: calculate r1, r2, C, A using Eqs. (2.3) and (2.4).
16: leader = closest leader(s) to X_i
17: leader = rand(leader,C)
18: if |A| < 1 then
19: N = leader - X_i
20: O = X_i - leader
21: D = |O|
22: step = |\{(1 - |A|) × D\}|
23: step = min(step, |O|, |N|)
24: X_i = X_i - rand(O, step)
25: X_i = X_i ∪ rand(N, step)
26: else
27: N = S_i - (X_i ∪ leader)
28: O = leader ∩ X_i
29: D = |X_i| - leader
30: step = (|A| - 1) × D
31: step = min(step, |O|, |N|)
32: X_i = X_i - rand(O, step)
33: S_i = rand(N, step)
34: X_i = X_i ∪ S_i
35: S_i = S_i - S_i
36: end if
37: end for
38: end for
39: return X_α

- BABC (Clodomir 2019) binary artificial bee colony.
- BFA (Bhattacharjee and Sarmah 2015) binary firefly algorithm.
- BCS (Kaya 2018) binary cuckoo search.

The experiments were carried out on an Asus Laptop with an Intel Core i3, 2.3 GHz processor, and 8GB memory working on Windows 10 OS using python programming language in VS code IDE. The source code of our algorithm is publicly available in (Roayaei 2020).

Three binary optimization problems (0/1 Knapsack, Vertex Cover, and Influence Maximization) are used as benchmark problems. For each problem, different datasets with different sizes are used in the experiments. The best value obtained from 30 independent runs is considered as the output of an algorithm. For each experiment, our results are compared with other versions of binary GWO, other binary optimizers, and optimal solutions or best-known solutions if exist.

For Vertex Cover, the optimal value for minimum vertex cover (MVC) of each problem instance is known (Da Silva et al. 2013). Thus, nine algorithms are compared with each other and with the optimal solution. The problem here is to find a subset of size k=MVC of vertices of the input graph such that it covers the maximum number of edges. MVC covers all edges. The characteristics of the datasets are shown in Table 2.

For 0/1 Knapsack, again, the optimal value (V*) is known (Ortega 2020). Thus, nine algorithms are compared with each other and with the optimal solution. The problem here is to find a subset of input elements such that the sum of the weight of its elements does not exceed the knapsack weight W, and the sum of the value of its elements is maximized. The characteristics of the datasets are shown in Table 3.

For Influence Maximization, the optimal value is not known for any problem instance. Thus, nine algorithms are compared with each other and with the best results obtained in the literature. The problem here is to find a k-subset of vertices of the input graph such that it maximizes the spread of influence. The characteristics of the datasets are shown in Table 4. The independent cascade (IC) is used as the information diffusion model, and the propagation probability, p, differs for each problem instance.

In the following, the proposed algorithm is compared with the other algorithms in terms of quality of solutions, running time, scalability, and convergence properties.

4.1 Quality of solutions

In this subsection, algorithms are compared with respect to the quality of solutions. For Vertex Cover and Knapsack, the population size is 100 and the number of iterations is 1000. For Influence Maximization, the population size and the number of iterations is 100. Since the running time of the fitness function for this problem is high, we decreased the number of iterations from 1000 to 100. The value of k is 30 for Ego-Facebook, and is 50 for other datasets. The results are shown in Table 5 to Table 7.
Table 3  Benchmark datasets for 0/1 Knapsack

| DataSet   | # Elements | W  | V*     | Ref.          |
|-----------|------------|----|--------|---------------|
| knapPI_1_100  | 100        | 995| 9147   | (Ortega 2020) |
| knapPI_1_200  | 200        | 1008| 11238  | (Ortega 2020) |
| knapPI_1_500  | 500        | 2543| 28857  | (Ortega 2020) |
| knapPI_1_1000 | 1000       | 5002| 54503  | (Ortega 2020) |
| knapPI_1_2000 | 2000       | 10011| 110635 | (Ortega 2020) |
| knapPI_1_5000 | 5000       | 25016| 276457 | (Ortega 2020) |

Table 4  Benchmark datasets for Influence Maximization

| DataSet | p  | # Nodes | # Edges | Ref.          |
|---------|----|---------|---------|---------------|
| HAM     | 0.03 | 2426    | 16631   | (Zareie et al. 2020) |
| Ego-Facebook | 0.01 | 4039    | 88234   | (Beni and Bouyer 2020) |
| Wiki-votes | 0.01 | 7115    | 103689  | (Bucur and Iacca 2016) |
| PGP     | 0.06 | 10680   | 24316   | (Arora et al. 2017) |
| Hepph   | 0.1  | 12008   | 118521  | (Arora et al. 2017) |
| NetHept | 0.1  | 15233   | 31376   | (Arora et al. 2017) |

Table 5  Quality comparison results on Vertex Cover

| DataSet       | Optimal | SetGWO | BCROSS | BPSO | BGA | BPROB | BGWO | ABC | BFA | BCS |
|---------------|---------|--------|--------|------|-----|-------|------|-----|-----|-----|
| CodeMiner     | 1109    | 994    | 992    | 837  | 930 | 798   | 889  | 845 | 796 | 944 |
| Airline       | 2101    | 2085   | 2096   | 2071 | 2078| 2029  | 2087 | 1882| 1876| 1991|
| USAir97       | 2126    | 2111   | 2109   | 2077 | 2087| 2027  | 2085 | 1858| 1752| 1942|
| CPAN authors  | 2248    | 2139   | 1748   | 1358 | 1528| 1524  | 1291 | 1736| 1719| 1912|
| EuroSiS       | 7586    | 7278   | 6604   | 5778 | 6053| 5235  | 6230 | 6040| 5822| 6068|
| YeastS        | 7182    | 6503   | 6483   | 5441 | 5705| 4654  | 6216 | 5657| 5137| 5977|

Table 6  Quality comparison results on 0/1 Knapsack

| DataSet         | Optimal | SetGWO | BCROSS | BPSO | BGA | BPROB | BGWO | ABC | BFA | BCS |
|-----------------|---------|--------|--------|------|-----|-------|------|-----|-----|-----|
| knapPI_1_100    | 9147    | 9147   | 9147   | 9147 | 9147| 8927  | 9147 | 9147| 9147| 9147|
| knapPI_1_200    | 11238   | 11238  | 11238  | 11056| 10704| 10444 | 11238|10615|10882|11040|
| knapPI_1_500    | 28857   | 28857  | 28655  | 28131| 26984| 25421 | 28241|26352|26386|28223|
| knapPI_1_1000   | 54503   | 53203  | 52197  | 51944| 41370| 40250 | 45680|49150|48464|51930|
| knapPI_1_2000   | 110625  | 107295 | 106777 | 101759|76772|74157  | 88432|93367|98080|103441|
| knapPI_1_5000   | 276457  | 267867 | 261892 | 219935|163179|159982 | 153179|215009|209557|240721|

Table 7  Quality comparison results on Influence Maximization

| DataSet       | Best Known | SetGWO | BCROSS | BPSO | BGA | BPROB | BGWO | BABBC | BFA | BCS |
|---------------|------------|--------|--------|------|-----|-------|------|-------|-----|-----|
| HAM           | 316        | 416    | 413    | 333  | 353 | 384   | 351  | 353   | 353 | 406 |
| Ego-Facebook  | 383        | 464    | 461    | 451  | 324 | 395   | 407  | 438   | 449 | 455 |
| Wiki-vote     | 265        | 240    | 219    | 122  | 161 | 169   | 124  | 174   | 186 | 208 |
| PGP           | 513        | 624    | 614    | 418  | 465 | 496   | 495  | 492   | 524 | 586 |
| Hepph         | 4410       | 4499   | 4148   | 4118 | 3995| 4070  | 4021 | 4103  | 3991| 4113|
| NetHept       | 980        | 1080   | 1055   | 702  | 840 | 1024  | 956  | 939   | 885 | 1020|
As can be seen, all algorithms find optimal or near-optimal solutions in small datasets. Expecting, as the complexity of a dataset increases, the quality of solutions of all algorithms decreases. The experimental results show that SetGWO performs the best on 16 out of 18 datasets. It obtained optimal or near-optimal solutions for most of the datasets of Vertex Cover and Knapsack, and better results than the best known solutions on five out of six datasets of Influence Maximization. The reason for better performance of SetGWO is that it uses a different distance measure, takes advantage of the inclusive exploration set $S_i$, and updates an omega with respect to one leader.

BCROSS, which uses crossover in its updating mechanism, performs the best on three and the second-best on 15 out of 18 datasets. It seems that the crossover operation performs well in discrete spaces for exploitation.

Some optimizers (e.g., BGWO) perform well on some problems (e.g., Knapsack), but perform poorly on other problems (e.g., Influence Maximization). Also, Some optimizers (e.g., BCROSS, which uses crossover in its updating mechanism, performs the best on three and the second-best on 15 out of 18 datasets. It seems that the crossover operation performs well in discrete spaces for exploitation.

Some optimizers (e.g., BCROSS and BGA) achieve better running time than the others. Since the runtime of BFA grows on the order of the square of the population size, it is much slower than the other algorithms, which makes it impractical for large optimization problems.

4.2 Running time

In this subsection, the running times of algorithms are compared. For each experiment, all parameters (population size, number of iterations, etc.) are the same. Results are shown in Figs. 4, 5 and 6.

The experimental results show that SetGWO achieves much better running time than all other algorithms on all datasets. The reason is that SetGWO uses no transfer function and boundary checking. Also, because the size of each agent is $k$ (the size of the solution) instead of $n$ (the number of elements), the updating process takes less time than the other algorithms.

As can be seen, algorithms that use binary values instead of real values and do not need boundary checking and transfer function (e.g., BCROSS and BGA) achieve better running time than the others. Since the runtime of BFA grows on the order of the square of the population size, it is much slower than the other algorithms, which makes it impractical for large optimization problems.

Note that the difference in running times of nine algorithms for Influence Maximization is less than the other two problems. The reason is that the fitness function of this problem takes much more time than that of the other two problems, and the majority of the execution time is spent on calculating fitness values for agents.

4.3 Scalability

While the quality of solutions and running time form important aspects of optimization techniques, scalability is an equally important aspect that determines the utility in practical scenarios. For an algorithm to be called scalable, it must scale well with both running time and the quality of solutions. In this subsection, the scalability of SetGWO are compared with the other algorithms on Knapsack datasets. The reason that Knapsack is selected for scalability comparison is that the optimal value for its all instance is known, and also, the complexity of an instance is dependent only on one factor (the number of elements); thus, we can analyze the extent of increase in error rate and running time based on the increase in the complexity.

We first analyze the scalability of SetGWO with respect to the quality of solutions. The error rate of an algorithm is calculated using Eq. (4.1):

$$\text{Error} = \frac{\text{Optimal} - \text{Solution}}{\text{Optimal}} \times 100$$

The results are shown in Fig. 7. Datasets are sorted according to their sizes. As can be seen, in less complex datasets, the error rate is low for all algorithms. As the size and complexity of datasets increase, the error rates also increase. Experimental results show that SetGWO is more scalable than other algorithms.

Scalability comparison results in terms of running time are shown in Fig. 8. Again, experimental results show that SetGWO is more scalable than other algorithms.

The reason for the better scalability of SetGWO is that it uses set operations, and each wolf is updated using a constant number of operations, regardless of the size of each solution. Also, the length of each wolf increases by the solution size, not the problem size. Furthermore, using an exclusive exploration set for each omega wolf, which is updated during the algorithm, makes it more scalable.

Again, among other versions of GWO, BCROSS, which uses crossover as its updating strategy, obtained the best scalability; and BGWO, and BPROB, which use real values, obtained the worst scalability.

Among non-GWO binary optimizers, BFA has the worst time scalability because its running time grows on the order of the square of the population size. For each pair of agents, the distance must be calculated, where the time of calculating the distance between two agents directly depends on the size of the dataset (which specifies the number of dimensions of an agent). Hence, the running time of BFA greatly increases with the increase in the size of datasets.
4.4 Convergence properties

To analyze the optimization behavior of the proposed algorithm, the convergence graphs for the two datasets knapPI_1_2000 and knapPI_1_5000 are shown in Fig. 9. For a better illustration, the results for each dataset are shown into two separate figures, both of which contain SetGWO results. As can be seen, BGWO, BPROB, and BGA reach their best fitness values in the early iterations and cannot improve their results in the following iterations. It seems that they lack a proper exploration strategy. The premature convergence proves that the algorithms spent most of the time in locally optimal solutions. The ability of global search in these algorithms on these datasets is considerably weak. After half
of the iterations, the explorative capability of the algorithms dropped dramatically.

SetGWO and BCROSS explore the solution space better than others, and their best fitness values increase with the increase in the number of iteration. Although SetGWO achieves the best fitness values, BCROSS achieves near-best values faster than SetGWO.

Also, the convergence graphs show that BPSO, BABC, BFA, and BCS achieve better solutions than SetGWO in the first half of the iterations, but cannot keep this superiority; SetGWO outperforms them eventually. BCS converges to near-best solutions, but does not manage to find the best solutions in the last part of the search.

Finally, it is found that SetGWO converges slowly and requires more iterations than the other algorithms to explore the solution space and achieve the best solution. It is well-capable of improving its solution steadily in the long run. It seems that this property directly originates from its exploration strategy and operators. On the other hand, algorithms that try to binarize the same operators as in their continuous version make the algorithm defenseless against the premature convergence and may result in poor exploratory.
In this paper, we reviewed different proposed algorithms dealing with the binarization of well-known GWO, which is a real coded optimizer. Based on the encoding scheme, updating strategy, and transfer function, they were classified into different categories.

Then, we proposed SetGWO, a novel algorithm based on set encoding and set updating strategy, as a binary version of GWO. SetGWO was compared with three other binary GWO algorithms, binary genetic algorithm, binary particle swarm intelligence, binary artificial bee colony, binary firefly algorithm, and binary cuckoo search, on 18 different datasets of three benchmark problems (Influence Maximization, Vertex Cover, and Knapsack). The results showed that our proposed algorithm outperforms other binary GWO algorithms with respect to the quality of solution, running time, and scalability.

The main strengths of SetGWO can be summarized as follows:

- Since it uses set encoding scheme, the size of each wolf is $k$ (the size of the solution) instead of $n$ (the number of elements); thus, the updating process takes less time than the other approaches which use other encoding schemes.
- Because of using set encoding scheme and set operators, there is no need to use transfer function and boundary checking, thus the running time is decreased.
- Because of using an exclusive exploration set for each agent, the exploration, and as a result, the quality of solution is improved.
- The newly defined distance measure helps the algorithm to better determine the distance of two solutions in a discrete space.
- The encircling strategy in SetGWO (updating the position of an omega according to its closest leader) helps the algorithm to do more meaningful exploitation in discrete spaces.

There are several future directions for this research. First, we will apply and analyze SetGWO on the feature selection problem, in which the goal is to find a subset of the original features of a dataset, such that an induction algorithm running on data containing only the selected features will generate a predictive model that has the highest possible accuracy. Second, we plan for parallelizing SetGWO.
to improve its scalability on larger datasets. Third, we will modify SetGWO to fit permutation problems. Last, as stated in the result section, although SetGWO achieves the best fitness values, it converges slowly and requires more iterations than BCROSS to achieve its best solution. We plan to make its optimization process faster such that it achieves the same results in fewer iterations.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Abdell-Basset M, El-Shahat D, El-Henawy I (2019) Solving 0–1 knapsack problem by binary flower pollination algorithm. Neural Comput Appl 31(9):5477–5495
Abualigah L, Yousri D, Abd Elaziz M, Ewees AA, Al-qaness MA, Gandomi AH (2021) Aquila optimizer: a novel Meta-Heuristic optimization algorithm. Comput Ind Eng 157:107250
Abualigah L, Diaba A (2021) Advances in sine cosine algorithm: a comprehensive survey. Artif Intell Rev 1–42
Abualigah L, Diabat A, Mirjalili S, Abd Elaziz M, Gandomi AH (2021) The arithmetic optimization algorithm. Comput Methods Appl Mech Eng 376:113609
Abualigah L (2019) Feature selection and enhanced krill herd algorithm for text document clustering. Springer, Berlin, pp 1–165
Ali-Tashi Q, Kadiri SJA, Rais HM, Mirjalili S, Alhussian H (2019) Binary optimization using hybrid grey wolf optimization for feature selection. IEEE Access 7:39496–39508
Ali-Tashi Q, Abdulkadir SJ, Rais HM, Mirjalili S, Alhussian H, Ragab MG, Alqubashia A (2020) Binary Multi-Objective Grey Wolf Optimizer for Feature Selection in Classification. IEEE Access 8:106247–106265
Arora A, Galhotra S, Ranu S (2017) Debunking the myths of influence maximization: An in-depth benchmarking study. In Proceedings of the 2017 ACM international conference on management of data (pp 651–666)
Bello R, Gomez Y, Nowe A, Garcia MM (2007) October. Two-step particle swarm optimization to solve the feature selection problem. In Seventh International Conference on Intelligent Systems Design and Applications (ISDA 2007) (pp 691–696). IEEE
Beni HA, Bouyer A (2020) TI-SC: top-k influential nodes selection based on community detection and scoring criteria in social networks. J Ambient Intell Humaniz Comput. https://doi.org/10.1007/s12652-020-01760-2
Bhattacharjee K K, Sarmah S P (2015) A binary firefly algorithm for knapsack problems. In 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM) (pp 73–77). IEEE
Boveiri HR, Khayami R, Elhoseny M, Gunasekaran M (2019) An efficient Swarm-Intelligence approach for task scheduling in cloud-based internet of things applications. J Ambient Intell Humaniz Comput 10(9):3469–3479
Bucur D, Iacca G (2016) Influence maximization in social networks with genetic algorithms. In European conference on the applications of evolutionary computation (pp 379–392). Springer, Cham
Cao D (2020) GraphMotifParameters. https://alg-git.informatik.uni-kl.de/Dai/GraphMotifParameters
Chantar H, Mafarja M, Alsawalqah H, Heidari AA, Aljarah I, Faris H (2020) Feature selection using binary grey wolf optimizer with elite-based crossover for Arabic text classification. Neural Comput Appl 32(16):12201–12220. https://doi.org/10.1007/s00521-019-04368-6
Da Silva MO, Gimenez-Lugo GA, Da Silva MV (2013) Vertex cover in complex networks. Int J Mod Phys C 24(11):1350078. https://doi.org/10.1142/S0129183113500782
Devanathan K, Ganapathy S, Swaminathan R (2019) Binary Grey Wolf Optimizer based Feature Selection for Nucleolar and Centromere Staining Pattern Classification in Indirect Immunofluorescence Images. In 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) (pp. 7040–7043)
El-Kenawy ESM, Eid MM, Saber M, Ibrahim A (2020) MhGWO-SFS: Modified binary grey wolf optimizer based on stochastic fractal search for feature selection. IEEE Access 8:107635–107649
El-Shafei E, Sallam KM, Chakrabortty RK, Abohonay AA (2021) A clustering based Swarm Intelligence optimization technique for the Internet of Medical Things. Exp Syst Appl 173:114648
Emary E, Zawhah HM, Hassanien AE (2016) Binary grey wolf optimization approaches for feature selection. Neurocomputing 172:371–381. https://doi.org/10.1016/j.neucom.2015.06.083
Faris H, Aljarah I, Al-Betar MA, Mirjalili S (2018) Grey wolf optimizer: a review of recent variants and applications. Neural Comput Appl 30(2):413–435. https://doi.org/10.1007/s00521-017-3272-5
Fomin F V, Kratsch D, Woeginger G J (2004) Exact (exponential) algorithms for the dominating set problem. In International Workshop on Graph-Theoretic Concepts in Computer Science (pp 245–256). Springer, Berlin, Heidelberg
Hu P, Pan JS, Chu SC (2020) Improved Binary Grey Wolf Optimizer and Its application for feature selection. Knowl-Based Syst. https://doi.org/10.1016/j.knosys.2020.105746
Kabir MM, Shahjahan M, Mursale K (2011) A new local search based hybrid genetic algorithm for feature selection. Neurocomputing 74(17):2914–2928
Katagiri H, Hayashida T, Nishizaki I, Guo Q (2012) A hybrid algorithm based on community detection and scoring criteria in social networks. In Seventh International Conference on Intelligent Systems Design and Applications (ISDA 2007) (pp 691–696). IEEE
Kavitha H, Hayashida T, Nishizaki I, Guo Q (2012) A hybrid algorithm based on tabu search and ant colony optimization for k-minimum spanning tree problems. Exp Syst Appl 39(5):5681–5686. https://doi.org/10.1016/j.eswa.2011.11.103
Kaya Y (2018) Feature selection using binary cuckoo search algorithm. In 2018 26th Signal Processing and Communications Applications Conference (SIU) (pp. 1–4). IEEE
Kornaki GM, Kavanfar V (2015) Grey Wolf Optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time. J Comput Sci 8:109–120. https://doi.org/10.1016/j.jocs.2015.03.011
Kumar A, Khorwal R, Chaudhary S (2016) A survey on sentiment analysis using swarm intelligence. Indian J Sci Technol 9(39):1–7
Liu J, Sun T, Luo Y, Yang S, Cao Y, Zhai J (2020) Echo state network optimization using binary grey wolf algorithm. Neurocomputing 381.https://doi.org/10.1016/j.neucom.2015.06.083
Luo K, Zhao Q (2019) A binary grey wolf optimizer for the multidimensional knapsack problem. Appl Soft Comput 83:105645. https://doi.org/10.1016/j.asoc.2019.105645
Long W, Cai S, Jiao J, Tang M (2020) An efficient and robust grey wolf optimizer algorithm for large-scale numerical optimization. Soft Comput 24(2):997–1026
Luo K, Zhao Q (2019) A binary grey wolf optimizer for the multidimensional knapsack problem. Appl Soft Comput 83:105645. https://doi.org/10.1016/j.asoc.2019.105645
Manikandan SP, Manimegalai R, Hariharan MJCSTT (2016) Gene Selection from microarray data using binary grey wolf algorithm for classifying acute leukemia. Curr Signal Transd Therap 11(2):76–83. https://doi.org/10.2174/15743624116661600700784415
M. Roayaei, Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007

Moll M (2018) GEPHI Datasets. https://github.com/gephi/gephi/wiki/Datasets

Ortega J (2020) 0/1 Knapsack Datasets. http://artemisa.unicauca.edu.co/~johnyortega/instances_01_KP

Rao RS, Malathi PJSC (2019) A novel PTS: grey wolf optimizer-based PAPR reduction technique in OFDM scheme for high-speed wireless applications. Soft Comput 23(8):2701–2712

Rebello G, de Oliveira EJ (2020) Modified Binary Grey Wolf Optimizer. In Frontier Applications of Nature Inspired Computation (pp 148-179). Springer, Singapore

Roayaei M (2020) SetGWO. https://github.com/mroayaei/SetGWO.git

Sahoo A, Chandra S (2017) Multi-objective grey wolf optimizer for improved cervix lesion classification. Appl Soft Comput 52:64–80. https://doi.org/10.1016/j.asoc.2016.12.022

Santana, CJ et al (2019) A novel binary artificial bee colony algorithm. Futur Genera Comput Syst 98:180–196

Schranz M, Di Caro GA, Schmickl T, Elmenreich W, Arvin F, Şekerçioğlu A, Sende M (2021) Swarm intelligence and cyber-physical systems: concepts, challenges and future trends. Swarm Evolut Comput 60:100762

Sharma M, Singh G, Singh R (2019) A review of different cost-based distributed query optimizers. Progress Artif Intell 8(1):45–62

Sharma M, Sharma S, Singh G (2020) Remote monitoring of physical and mental state of 2019-nCoV victims using social internet of things, fog and soft computing techniques. Computer methods and programs in biomedicine 196:105609

Sunto DS, Ayon SI, Akhand MAH, Siddique N (2018) Modified Grey Wolf Optimization to Solve Traveling Salesman Problem. In 2018 International Conference on Innovation in Engineering and Technology (ICIET) (pp. 1-4)

Tu Q, Chen X, Liu X (2019) Multi-strategy ensemble grey wolf optimizer and its application to feature selection. Appl Soft Comput 76:16–30. https://doi.org/10.1016/j.asoc.2018.11.047

Zareie A, Sheikhahmadi A, Jalili M (2020) Identification of influential users in social network using gray wolf optimization algorithm. Exp Syst Appl 142:112971. https://doi.org/10.1016/j.eswa.2019.112971

Zemmam N, Azizi N, Sellami M, Cheriguene S, Ziani A, AlDwairi M, Dendani N (2020) Particle swarm optimization based swarm intelligence for active learning improvement: Application on medical data classification. Cognit Comput 12(5):991–1010

Zhang S, Zhou Y, Li Z, Pan W (2016) Grey wolf optimizer for unmanned combat aerial vehicle path planning. Adv Eng Softw 99:121–136. https://doi.org/10.1016/j.advengsoft.2016.05.015

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.