On the Binarization of Grey Wolf Optimizer: A Novel Binary Optimizer Algorithm

Mehdy Roayaei (mroayaei@modares.ac.ir)
Tarbiat Modares University https://orcid.org/0000-0001-9843-5886

Research Article

Keywords: Grey Wolf Optimizer, Binary Combinatorial Optimization, Evolutionary algorithm, Metaheuristics

Posted Date: May 7th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-235817/v1

Noname manuscript No.
(will be inserted by the editor)

On the Binarization of Grey Wolf Optimizer: A Novel Binary Optimizer Algorithm

Mehdy Roayaei

Received: date / Accepted: date

Abstract Grey Wolf Optimizer (GWO) is a population-based evolutionary algorithm inspired by the hunting behaviour of grey wolves. GWO, in its basic form, is a real coded algorithm, therefore, it needs modifications to deal with binary optimization problems. In this paper, we review previous works on binarization of GWO, and classify them 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. Experimental results on different real-world combinatorial optimization problems and different 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 · Evolutionary algorithm · 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), and 0/1 Knapsack (Abdel-Basset et al., 2019) are a few to mention.

Mehdy Roayaei
Corresponding author
Tel.: +98-21-82883939
E-mail: mroayaei@modares.ac.ir
Department of Electrical and Computer Engineering, Tarbiat Modares University, Tehran, Iran

Metaheuristics are general algorithmic frameworks designed to solve complex optimization problems. Grey wolf optimizer (GWO) (Mirjalili et al., 2014) is one of the evolutionary 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 the exploration and exploitation during the search which leads to favorable convergence.

GWO in its basic form is a real coded algorithm and therefore, it needs modifications to deal with binary optimization problems. There have been many works in the literature on binarization of GWO in recent years. In this paper, we review binary variations of GWO 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 different datasets (18 datasets), show that SetGWO outperforms other existing binary GWO algorithms in terms of quality of solutions, running time, and scalability.

In the remainder of the paper, the set of all input elements from which we are trying to select a subset is denoted by $\mathcal{S}$, where $|\mathcal{S}| = n$. The parameter k denotes the maximum possible size of a selected subset. Also,

k-set denotes a set of size k. Furthermore, rand($\mathcal{S}$,k) denotes a random k-subset of $\mathcal{S}$.

The remainder of the paper is organized as follows. In Section 2, the basic GWO is introduced. In Section 3, related work on binarization of the basic GWO is reviewd. The proposed binary version of GWO is described in section 4. The experimental results are discussed in section 5. Finally, conclusions are stated in section 6.

2 An Overview of the basic GWO

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(α), beta(β), delta(δ), and omega(ω). The best three grey wolves are considered as alpha, beta, and delta, and the remaining grey wolves are termed as omega. GWO simulates the major steps of grey wolves hunting, searching for prey, encircling, and attacking. The hunting is guided by α, β, and δ. The ω 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 α, β, and δ, respectively. Other wolves (ω 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 (new α, β, and δ) and the next iteration starts. The algorithm goes on until a condition of termination is reached.

The Eqs. (2.1) and (2.2) mathematically model encircling behavior of GWO:

$$\vec{D} = |\vec{C}.\vec{X}_p(t) - \vec{X}(t)| \tag{2.1}$$

$$\vec{X}(t+1) = \vec{X}_p(t) - \vec{A}.\vec{D} \tag{2.2}$$

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

$$\vec{A} = 2\vec{a}.\vec{r}_1 - \vec{a} \tag{2.3}$$

$$\vec{C} = 2.\vec{r}_2 \tag{2.4}$$

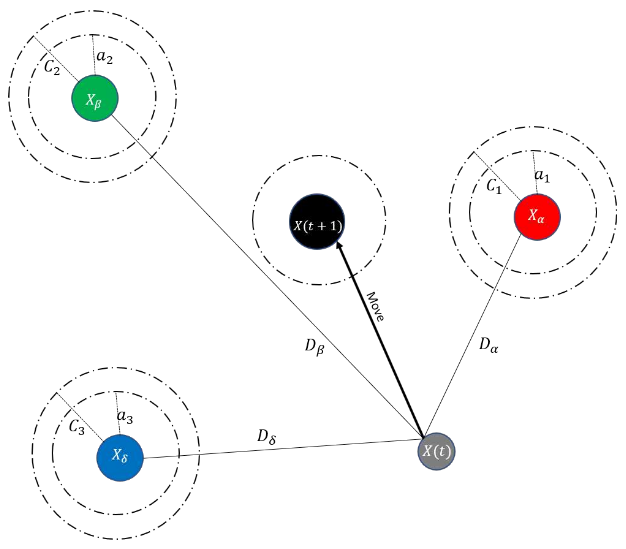


Fig. 1 Encircling Strategy in GWO

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 $\vec{C}$ provides random weights for prey in order to stochastically emphasize ($\vec{C} > 1$) or deemphasize ($\vec{C} < 1$) the effect of prey in defining the distance in Eq. (2.1).

Since we do not know the position of the prey (optimal solution) 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):

$$\vec{X}(t+1) = \frac{\vec{X}_1(t) + \vec{X}_2(t) + \vec{X}_3(t)}{3}, \tag{2.5}$$

where X_1, X_2, and X_3 are calculated using Eqs. (2.1) and (2.2), considering that $\vec{X}_p$ is replaced with $\vec{X}_\alpha$, $\vec{X}_\beta$, and $\vec{X}_\delta$ respectively. Fig. 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 et al., 2015), Vehicle Path Planning (Zhang et al., 2016), Feature Selection (Al-Tashi et al., 2020), Multidimensional Knapsack (Luo et al., 2019), Numeric Optimization (Long et al. , 2020), Traveling Salesman Problem (Sopto et al., 2018), Signal Processing (Rao et al. , 2019), and Text Classification (Chantar et al., 2020) are just a few to mention.

3 Related Work on Binarization of GWO

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. Binary optimization problems use binary space, in which the solution is limited to 0 and 1 values, which becomes a problem for the typical GWO. Hence, there have been many efforts in recent years to modify the basic GWO for binary optimization problem.

Each binarization approach for GWO must consider three important components. The first component is **encoding scheme**, which must propose a scheme for encoding solutions of a binary optimization problem in 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, we review related work on binarization of GWO with respect to these three components.

One of the most cited work 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 gray wolf position. In the next 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 for choosing optimal elements subsets (Manikandan et al., 2016). In their approach, in which they used the binary encoding for solutions, 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 for the cervix lesion classification problem (Sahoo et al., 2017). They used binary encoding. As a transfer function, two steps were proposed to have binary representation of each grey wolf position. First, the $tanh()$ function scales the real coded position values to the range of $[0, 1]$. Then each scaled value is compared with a randomly generated threshold $T \in rand(0, 1)$. If it exceeds the threshold value, then the bit is set to 1 else 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 the 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 the binary encoding scheme. For the transfer function, they used a sigmoid 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 multidimensional 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, a new evolutionary mechanism with a differentiated position updating strategy (Luo et al., 2019). They used the binary representation for encoding each solution. For converting continuous values to binary values, they experimentally evaluated six different transfer functions and concluded that the absolute function of hyperbolic tangent among six transform functions performs better than others.

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

Zareie et al. proposed a probabilistic encoding scheme, in which, each solution (wolf) X_j is shown as a vector of n elements, where the 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 binary GWO based on Stochastic Fractal Search (SFS) to balance the 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 increase the search space and the exploitation, and the crossover/mutation operations to increase the diversity of the population to enhance exploitation capability. Then, 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 the proposed algorithm are then converted into binary values using a sigmoid function so that it can be used for the desired binary problem.

In (Rebello et al., 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 the algorithm to best suit its application to escape local minima.

There have been many modifications on the basic GWO in the previous work. However, considering the encoding scheme, updating strategy, and transfer function, the 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 $\in [0\ 1]$, which indicates the probability of selection of that 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 poisition of an omega wolf.

Also, different transfer fucntions such as v-shape, sigmoid, threshold and tanh() are used in previous work to convert real numbers to 0 or 1 value.

Considering the basic GWO and previous work on binarization of it, we can summarize their disadvantages as follows:

- In all previous algorithms, each wolf is represented by a list 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. The 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 an infeasible solution. Thus, it seems that the encircling strategy must be modified.
- In exploitation (exploration), the distance of an omega wolf from the average position of leaders is decreased (increased). Because a real value can not 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 the exploration and exploitation strategy must be modified.

4 Proposed Algorithm

In this section, we propose SetGWO, a novel binary optimizer based on the basic GWO . In this algorithm, we use set encoding, where each wolf is represented as a set (a k-subset of $\mathcal{S}$). For updating strategy, we use only simple set operators (union, intersection, and difference). Because of using such encoding and updating strategy, there is no need to use a transfer function. We use the same parameters (C, A, r_1, r_2) as in the basic GWO. The flowchart of the proposed algorithm is shown in Fig. 2.

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

4.1 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 $\mathcal{S}$.

4.2 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 an infeasible solution in a discrete space. In SetGWO, omegas are updated according to one leader (closest leader to the omega) as in Fig. 3.

Table 1 Summarization of previous work on binary GWO

Author	Encoding Scheme	Updating Strategy	Transfer function	Ref.
Emary et al.	Binary	Crossover	-	(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 et al., 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 et al., 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	-	(Zareie et al., 2020)
El-kenawy et al.	Binary	Crossover	Sigmoid	(El-Kenawy et al., 2020)
Rebello et al.	Binary	Arithmetic	Sigmoid	(Rebello et al., 2020)

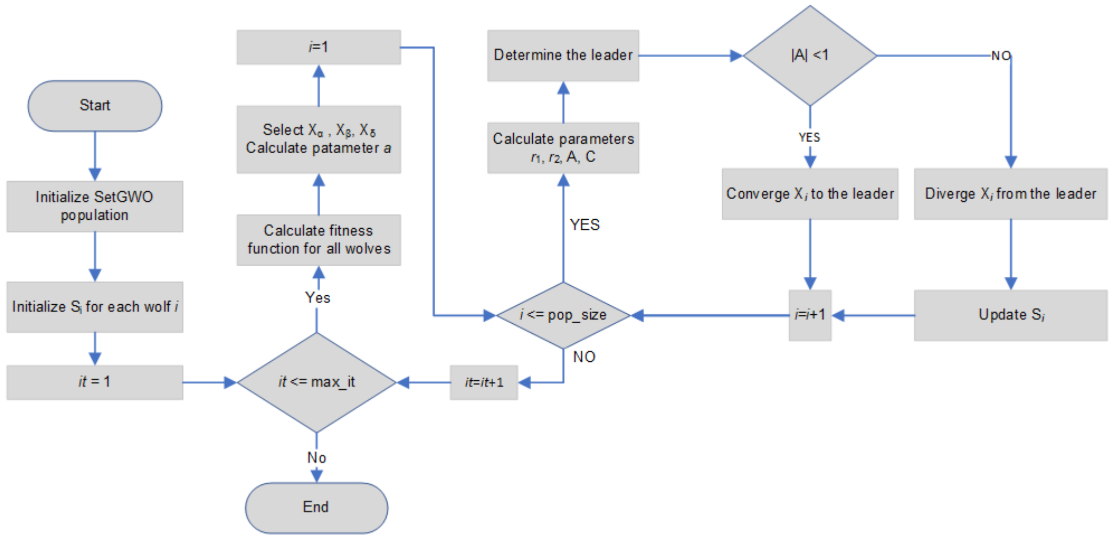

Start
Initialize SetGWO population
Initialize S_i for each wolf i
it = 1
it <= max_it
No
End
Yes
Calculate fitness function for all wolves
Select X_α, X_β, X_δ Calculate patameter a
i=1
i <= pop_size
NO
it=it+1
YES
Calculate parameters r1, r2, A, C
Determine the leader
|A| <1
YES
NO
Converge X_i to the leader
Diverge X_i from the leader
i=i+1
Update S_i

Fig. 2 Flowchart of SetGWO

4.3 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, we re-define the distance function for our algorithm. 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. (4.1):

$$D(X_i, leader) = X_i - leader \tag{4.1}$$

4.4 Updating Strategy

Since omegas are represented as sets, we use simple set operations (union, intersection, and difference) for updating an omega. As in the basic GWO, if $|A| < 1$, omega converges towards its closest leader, and if $|A| > 1$, omega diverges from its closest leader to hopefully find a better prey.

Exploitation
In exploitation, to decrease the distance of an omega from a leader, we substitute some elements of X_i that are not in the leader with the same number of elements of leader that are not in X_i, as in Eq. (4.2).

$$\begin{aligned} &\mathcal{N} = leader - X_i \\ &\mathcal{O} = X_i - leader \\ &D = |\mathcal{O}| \\ &step = \lceil |A| \times D \rceil \\ &X_i = X_i - rand(\mathcal{O}, step) \\ &X_i = X_i \cup rand(\mathcal{N}, step) \end{aligned} \tag{4.2}$$

where $\mathcal{N}$ is the set of elements of the leader that are not in X_i, $\mathcal{O}$ 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 *step* is the extent of which the distance D is decreased after convergence.

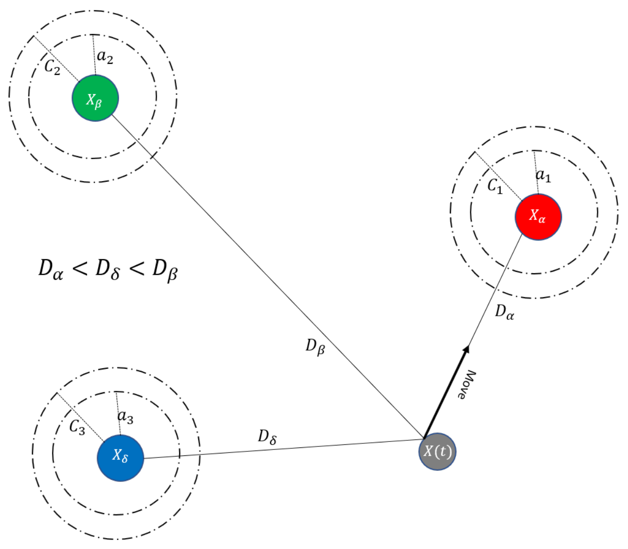


Fig. 3 Encircling strategy in SetGWO

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

$$\begin{aligned}
&\mathcal{N} = \mathcal{S}_i - (X_i \cup leader)\\
&\mathcal{O} = leader \cap X_i\\
&D = |X_i - leader|\\
&step = \lceil |A| \times D \rceil\\
&X_i = X_i - rand(\mathcal{O}, step)\\
&\mathcal{S}_{step} = rand(\mathcal{N}, step)\\
&X_i = X_i \cup \mathcal{S}_{step}\\
&\mathcal{S}_i = \mathcal{S}_i - \mathcal{S}_{step}
\end{aligned} \tag{4.3}$$

where $\mathcal{N}$ is the set of elements of $\mathcal{S}_i$ which are neither in X_i nor in the leader, $\mathcal{O}$ is the set of elements which are in both X_i and leader, D is the distance of the omega from the leader, *step* is the extent of which the distance D is increased after divergence, and $\mathcal{S}_{step}$ 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**.

5 Result and Discussion

In this section, SetGWO is benchmarked on three binary optimization problems and 18 datasets. To evaluate the proposed algorithm, we implemented three different binary versions of GWO, 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, we have implemented two other non-GWO binary opimizer algorithms to compare with:

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

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 availabe 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 five independent runs is considered as the output of an algorithm. For

Algorithm SetGWO

```
    Input: $\mathcal{S}$: 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 $\mathcal{S}$
 2: for $1 \le i \le pop\_size$ do
 3:     $\mathcal{S}_i = \mathcal{S}$
 4: end for
 5: for $1 \le it \le max\_it$ do
 6:     calculate the fitness of all wolves
 7:     $X_\alpha$ = the best wolf
 8:     $X_\beta$ = the second best wolf
 9:     $X_\delta$ = the third best wolf
10:     $a = 2 - it \times (\frac{2}{max\_it})$
11:     for $1 \le i \le pop\_size$ do
12:         if $(|\mathcal{S}_i| < t \times k)$ then
13:             $\mathcal{S}_i = \mathcal{S}$
14:         end if
15:         calculate $r_1$, $r_2$, 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:             $\mathcal{N}$ = leader - $X_i$
20:             $\mathcal{O}$ = $X_i$ - leader
21:             D = $|\mathcal{O}|$
22:             $step = \lceil |A| \times D \rceil$
23:             $step = min(step, |\mathcal{O}|, |\mathcal{N}|)$
24:             $X_i = X_i$ - rand($\mathcal{O}$,$step$)
25:             $X_i = X_i \cup$ rand($\mathcal{N}$,$step$)
26:         else
27:             $\mathcal{N} = \mathcal{S}_i$ - $(X_i \cup leader)$
28:             $\mathcal{O} = leader \cap X_i$
29:             D = —$X_i$ - leader—
30:             $step = \lceil |A| \times D \rceil$
31:             $step = min(step, |\mathcal{O}|, |\mathcal{N}|)$
32:             $X_i = X_i$ - rand($\mathcal{O}$,$step$)
33:             $\mathcal{S}_{step}$ = rand($\mathcal{N}$,$step$)
34:             $X_i = X_i \cup \mathcal{S}_{step}$
35:             $\mathcal{S}_i = \mathcal{S}_i$ - $\mathcal{S}_{step}$
36:         end if
37:     end for
38: end for
39: return $X_\alpha$
```

each experiment, we have compared our results with other versions of binary GWO, binary GA, binary PSO, and optimal solutions or best-knowns solution 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, we have compared six algorithms with each other and with the optimal solution. The problem here is to find a subset of size k=—MVC— of vertices of 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, we have compared six algorithms 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.

Table 2 Benchmark datasets for Vertex Cover

	DataSet	# Nodes	# Edges	MVC	Ref.
1	Airline	235	2101	96	(Moll, 2018)
2	US Air97	332	2126	149	(Moll, 2018)
3	Codeminer	724	1109	191	(Moll, 2018)
4	CPAN authors	839	2248	116	(Moll, 2018)
5	EuroSiS	1285	7586	597	(Moll, 2018)
6	YeastS	2361	7182	763	(Moll, 2018)

Table 3 Benchmark datasets for 0/1 Knapsack

	DataSet	# Elements	W	V^*	Ref.
1	knapPI_1_100	100	995	9147	(Ortega, 2020)
2	knapPI_1_200	200	1008	11238	(Ortega, 2020)
3	knapPI_1_500	500	2543	28857	(Ortega, 2020)
4	knapPI_1_1000	1000	5002	54503	(Ortega, 2020)
5	knapPI_1_2000	2000	10011	110635	(Ortega, 2020)
6	knapPI_1_5000	5000	25016	276457	(Ortega, 2020)

For Influence Maximization, the optimal value is not known for any problem instance. Thus, we have compared six algorithms 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.

Table 4 Benchmark datasets for Influence Maximization

	DataSet	p	# Nodes	# Edges	Ref.
1	HAM	0.03	2426	16631	(Zareie et al., 2020)
2	Ego-Facebook	0.01	4039	88234	(Beni et al., 2020)
3	Wiki-votes	0.01	7115	103689	(Bucur et al. , 2016)
4	PGP	0.06	10680	24316	(Arora et al., 2017)
5	Hepph	0.1	12008	118521	(Arora et al., 2017)
6	NetHept	0.1	15233	31376	(Arora et al., 2017)

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

5.1 Quality of solutions

In this subsection, algorithms are compared with respect to 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.

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 obtained better or equal results than others five algorithms in 16 out of 18 benchmark 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 for 5 out of 6 datasets of Influence Maximization. The better quality of solutions of SetGWO is because of using a more reasonable distance measure, the inclusive exploration set $\mathcal{S}_i$ which improves the exploration, and updating an omega with respect to one leader. Among other five algorithms, BCROSS, which uses crossover as its updating mechanism, obtained best results.

5.2 Running Time

In this subsection, we compare the running times of gorithms. For each experiment, all parameters (the number of population, number of iteration, etc.) are the same. Results are shown in Fig. 4 to Fig. 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 wolf 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, the algorithms that use integer values instead of real values and do not need boundary checking and transfer function (that is BCROSS and BGA) achieve much better running time than the others.

Note that the difference in running times of six 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 wolves.

5.3 Scalability

While 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 quality of solutions. In this subsection, we compare the scalability of SetGWO with the other algorithms on Knapsack datasets. The reason that Knapsack is selected for scalability comparison is that 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 trace the increasing of error rate and running time with respect to the increasing of the complexity.

We first analyze the scalability of SetGWO with respect to quality of solutions. The error rate of an algorithm is calculated using Eq. (5.1):

$$Error = \frac{Optimal - Solution}{Optimal} \times 100 \tag{5.1}$$

Result is 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 result in terms of running time is shown in Fig. 8. Datasets are sorted according to their sizes. Again, experimental results show that SetGWO is more scalable than other algorithms.

The reason for better scalability of SetGWO is that becuase of using set operations, updating each wolf is done 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 than the others. Again, among other versions of GWO, BCROSS, which uses crossover as it updating strategy, obtained best scalability; and, BGWO and BPROB, which use real values obtained worst scalability.

6 Conclusion and Future Work

In this paper, we reviewed different proposed algorithms dealing with binarization of well-known GWO, which is a real coded optimizer. Based on the encoding scheme, updating strategy, and transfer function, we classified them into different categories.

Then, we proposed SetGWO, a novel algorithm based on set encoding and set updating strategy, for binarization of GWO. We compared SetGWO with the other three binary GWO algorithms, binary GA, and binary

Table 5 Quality comparison results on Vertex Cover

DataSet	Optimal	SetGWO	BCROSS	BPSO	BGA	BPROB	BGWO
CodeMiner	1109	**994**	992	837	930	798	889
Airline	2101	2085	**2096**	2071	2078	2029	2087
USAir97	2126	**2111**	2109	2077	2087	2027	2085
CPAN authors	2248	**2139**	1748	1358	1528	1524	1291
EuroSiS	7586	**7278**	6604	5778	6053	5235	6230
YeastS	7182	**6503**	6483	5441	5705	4654	6216

Table 6 Quality comparison results on 0/1 Knapsack

DataSet	Optimal	SetGWO	BCROSS	BPSO	BGA	BPROB	BGWO
knapPI_1_100	9147	**9147**	**9147**	**9147**	**9147**	**9147**	**9147**
knapPI_1_200	11238	**11238**	**11238**	11056	10704	10444	**11238**
knapPI_1_500	28857	**28857**	28655	28131	26984	25421	28241
knapPI_1_1000	54503	**53203**	52197	51944	41370	40250	45680
knapPI_1_2000	110625	**107295**	106777	101759	76772	74157	88432
knapPI_1_5000	276457	**267867**	261892	219935	163179	159982	153179

Table 7 Quality comparison results on Influence Maximization

DataSet	Best Known	SetGWO	BCROSS	BPSO	BGA	BPROB	BGWO
HAM	316	**416**	415	351	384	353	333
Ego-Faceboob	383	**464**	461	407	395	324	451
Wiki-vote	**265**	240	219	124	169	161	122
PGP	513	**624**	614	495	496	465	418
Hepph	4410	**4499**	4148	4021	4070	3995	4118
NetHept	980	**1080**	1055	956	1024	840	702

PSO on 18 different datasets of three benchmark problems (Infleuence Maximization, Vertex Cover, and Knapsack) and showed that our proposed algorithm, SetGWO, outperforms other binary GWO algorithms with respect to quality of solution, running time and scalability.

The main advantages of SetGWO can be summarized as follows:

- Since it uses the 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, which decreases the running time.
- Because of using an exclusive exploration set for each omega, denoted by $\mathcal{S}_i$, 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 Feature Selection Problem, in which the goal is finding 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 very large datasets. And, third, we will modify SetGWO in order to fit for permutation problems.

Note that in this paper, we have focused on different binary variants of GWO, and thus we have compared our results with only two other non-GWO algorithms (binary GA and binary PSO). In the future work, we plan to compare the perfarmance of SetGWO with other state-of-the-art algorithms.

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.

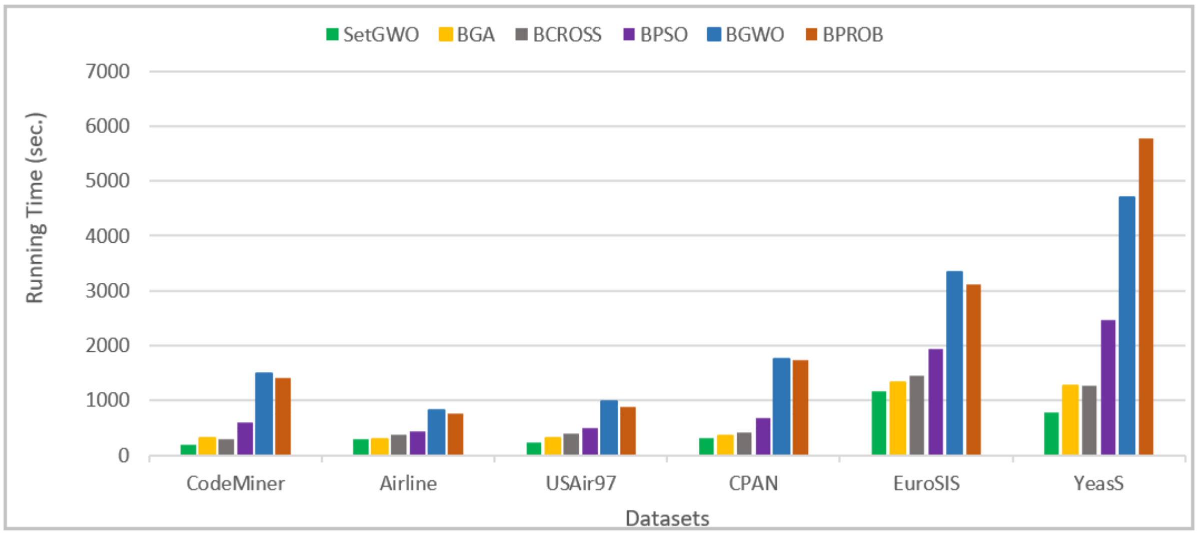


Fig. 4 Running time comparison results on Vertex Cover

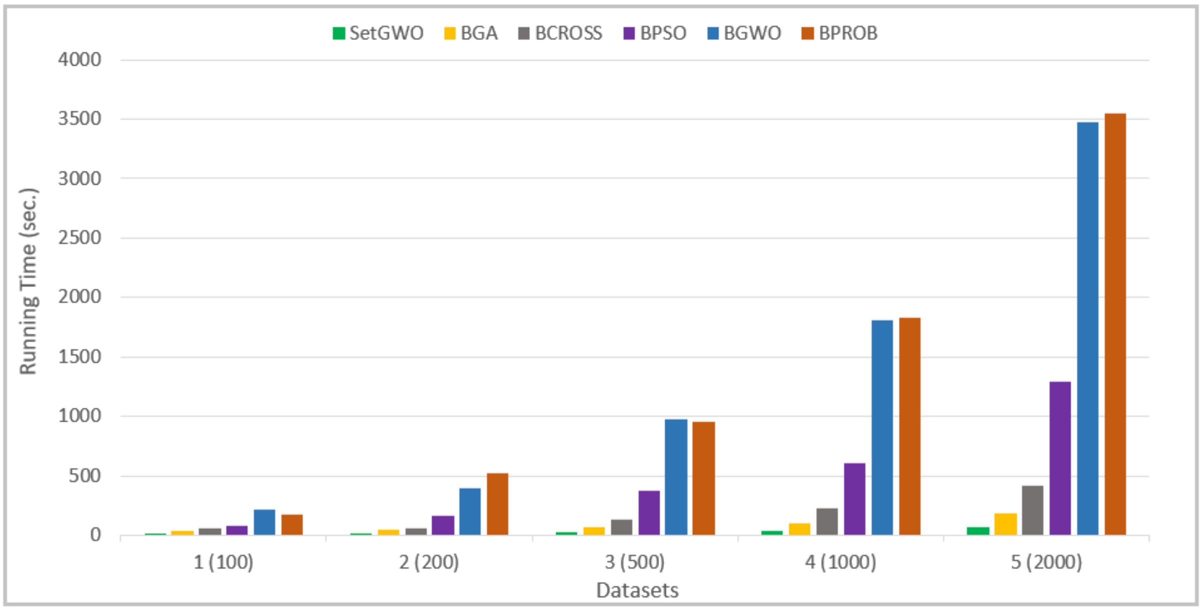


Fig. 5 Running time comparison results on Knapsack

References

Abdel-Basset, M., El-Shahat, D. and El-Henawy, I., 2019. Solving 0–1 knapsack problem by binary flower pollination algorithm. *Neural Computing and Applications*, 31(9), pp.5477-5495.

Al-Tashi, Q., Kadir, S. J. A., Rais, H. M., Mirjalili, S., & Alhussian, H. (2019). Binary optimization using hybrid grey wolf optimization for feature selection. *IEEE Access*, 7, 39496-39508.

Al-Tashi, Q., Abdulkadir, S. J., Rais, H. M., Mirjalili, S., Alhussian, H., Ragab, M. G., & Alqushaibi, A. (2020). Binary Multi-Objective Grey Wolf Optimizer for Feature Selection in Classification. *IEEE Access*, 8, 106247-106263.

Arora, A., Galhotra, S., & Ranu, S. (2017, May). 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. and Garcia, M.M., 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, H. A., & Bouyer, A. (2020). TI-SC: top-k influential nodes selection based on community detection and scoring criteria in social networks. *Jour-*

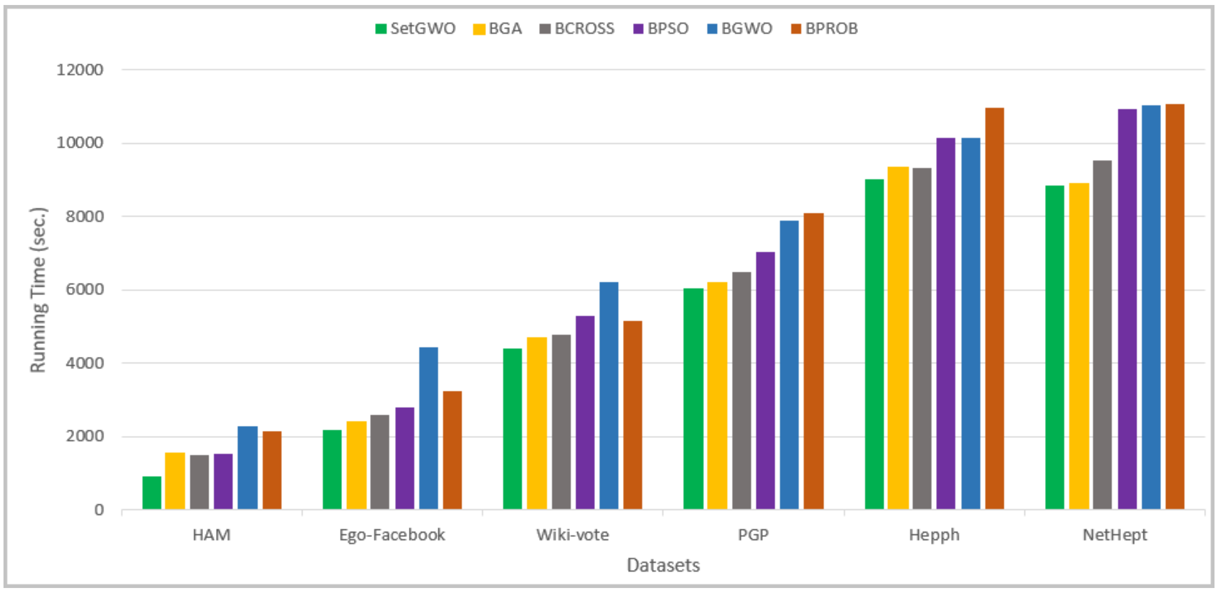


Fig. 6 Running time comparison results on Influence Maximization

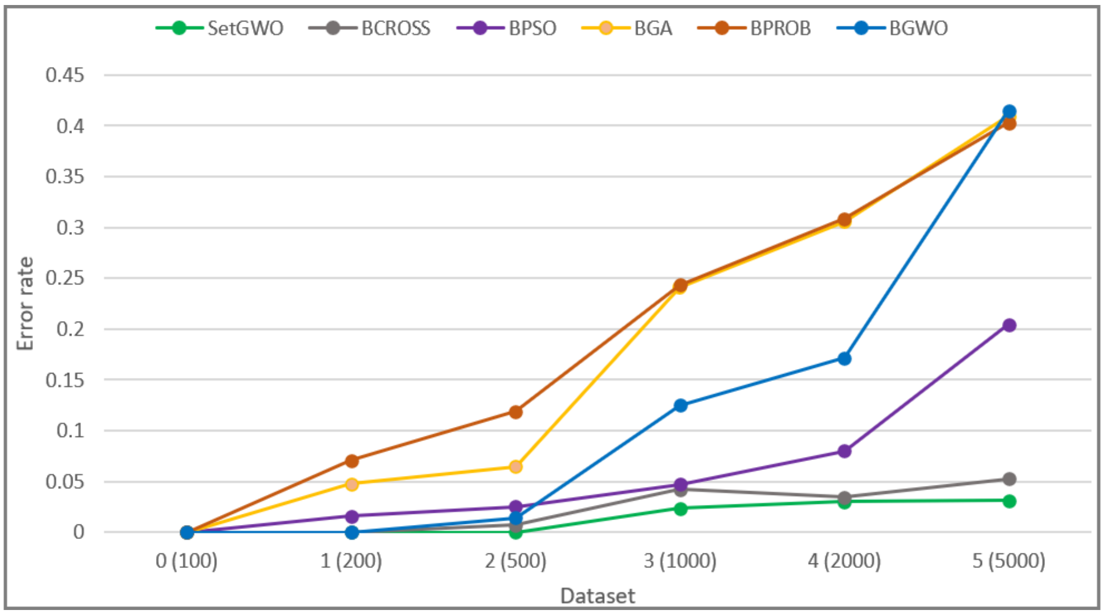


Fig. 7 Scalability comparison results on Knapsack in terms of quality of solutions

nal of Ambient Intelligence and Humanized Computing, 1-20. https://doi.org/10.1007/s12652-020-01760-2.

Bucur, D., & Iacca, G. (2016, March). 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, A. A., Aljarah, I., & Faris, H. (2020). Feature selection using binary grey wolf optimizer with elite-based crossover for Arabic text classification. *Neural Computing and Applications*, 32(16), 12201-12220. https://doi.org10.1007/s00521-019-04368-6.

Da Silva, M. O., Gimenez-Lugo, G. A., & Da Silva, M. V. (2013). Vertex cover in complex networks. *International Journal of Modern Physics C*, 24(11), 1350078. https://doi.org/10.1142/S0129183113500782.

Devanathan, K., Ganapathy, N., & Swaminathan, R. (2019, July). Binary Grey Wolf Optimizer based Feature Selection for Nucleolar and Centromere Stain-

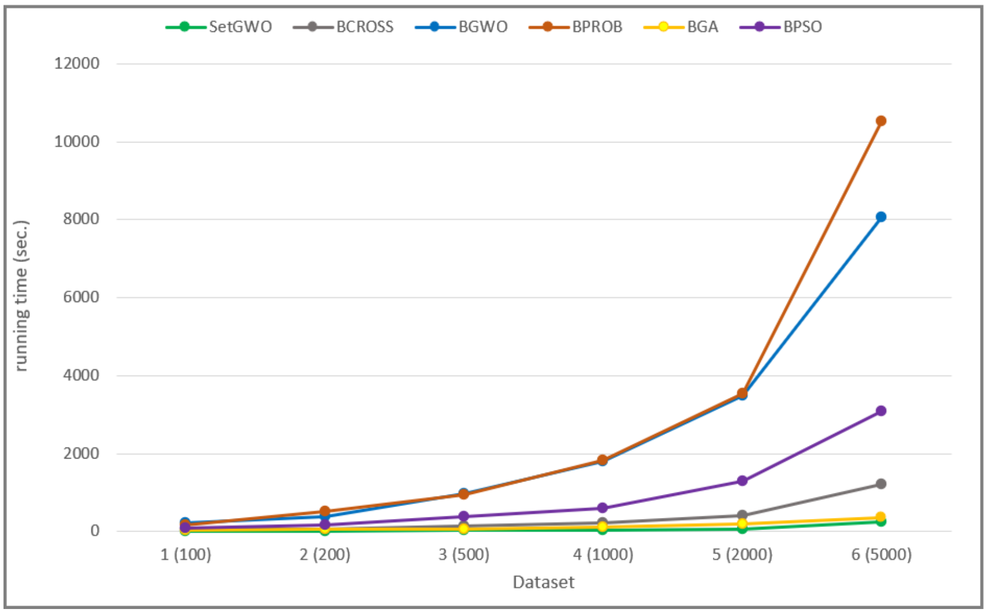


Fig. 8 Scalability comparison results on Knapsack in terms of running time

ing 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, E. S. M., Eid, M. M., Saber, M., & Ibrahim, A. (2020). MbGWO-SFS: Modified binary grey wolf optimizer based on stochastic fractal search for feature selection. *IEEE Access*, 8, 107635-107649.

Emary, E., Zawbaa, H. M., & Hassanien, A. E. (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, M. A., & Mirjalili, S. (2018). Grey wolf optimizer: a review of recent variants and applications. *Neural computing and applications*, 30(2), 413-435. `https://doi.org/10.1007/s00521-017-3272-5`.

Fomin, F. V., Kratsch, D., & Woeginger, G. J. (2004, June). 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, J. S., & Chu, S. C. (2020). Improved Binary Grey Wolf Optimizer and Its application for feature selection. *Knowledge-Based Systems*, 105746. `https://doi.org/10.1016/j.knosys.2020.105746`.

Kabir, M.M., Shahjahan, M. and Murase, K., 2011. A new local search based hybrid genetic algorithm for feature selection. *Neurocomputing*, 74(17), pp.2914-2928.

Katagiri, 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. *Expert Systems with Applications*, 39(5), 5681-5686. `https://doi.org/10.1016/j.eswa.2011.11.103`.

Komaki, G. M., & Kayvanfar, V. (2015). Grey Wolf Optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time. *Journal of Computational Science*, 8, 109-120. `https://doi.org/10.1016/j.jocs.2015.03.011`.

Liu, J., Sun, T., Luo, Y., Yang, S., Cao, Y., & Zhai, J. (2020). Echo state network optimization using binary grey wolf algorithm. *Neurocomputing*, 385, 310-318. `https://doi.org/10.1016/j.neucom.2019.12.069`

Long, W., Cai, S., Jiao, J., & Tang, M. (2020). An efficient and robust grey wolf optimizer algorithm for large-scale numerical optimization. *Soft Computing*, 24(2), 997-1026

Luo, K., & Zhao, Q. (2019). A binary grey wolf optimizer for the multidimensional knapsack problem. *Applied Soft Computing*, 83, 105645. `https://doi.org/10.1016/j.asoc.2019.105645`

Manikandan, S. P., Manimegalai, R., & Hariharan, M. J. C. S. T. T. (2016). Gene Selection from microarray data using binary grey wolf algorithm for classifying acute leukemia. *Current Signal Transduction Therapy*, 11(2), 76-83. `http://dx.doi.org/10.2174/1574362411666160607084415`

Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. *Advances in engineering software*, 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, R. S., & Malathi, P. J. S. C. (2019). A novel PTS: grey wolf optimizer-based PAPR reduction technique in OFDM scheme for high-speed wireless applications. *Soft Computing*, 23(8), 2701-2712

Rebello, G., & de Oliveira, E. J. (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. *Applied Soft Computing*, 52, 64-80. `https://doi.org/10.1016/j.asoc.2016.12.022`.

Sopto, D. S., Ayon, S. I., Akhand, M. A. H., & Siddique, N. (2018, December). 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. *Applied Soft Computing*, 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. *Expert Systems with Applications*, 142, 112971. `https://doi.org/10.1016/j.eswa.2019.112971`.

Zhang, S., Zhou, Y., Li, Z., & Pan, W. (2016). Grey wolf optimizer for unmanned combat aerial vehicle path planning. *Advances in Engineering Software*, 99, 121-136. `https://doi.org/10.1016/j.advengsoft.2016.05.015`

Figures

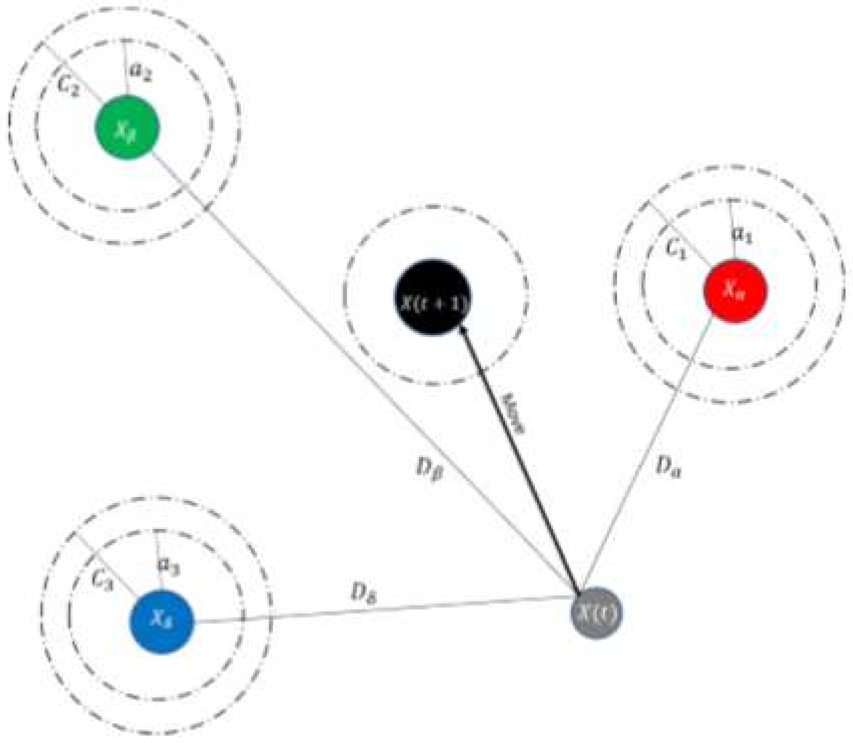
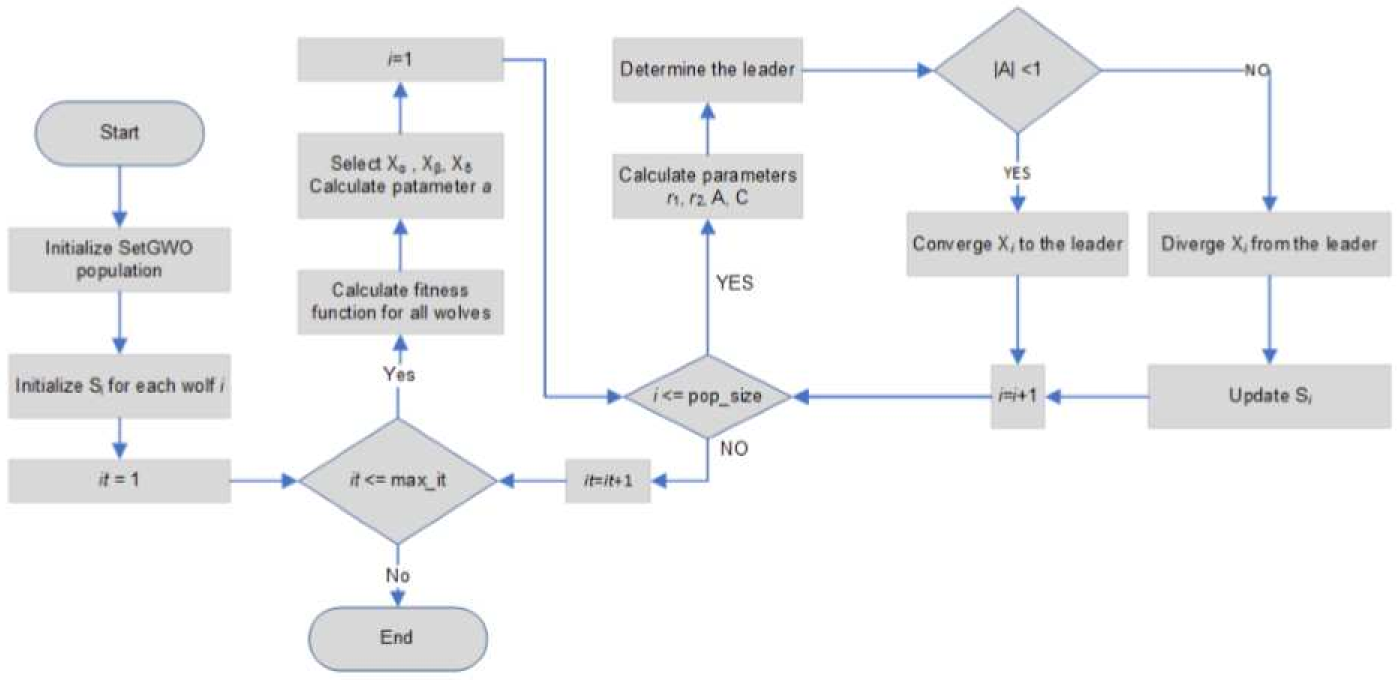


Figure 1

Encircling Strategy in GWO



Start
Initialize SetGWO population
Initialize S_i for each wolf i
$it = 1$
$it <= max\_it$
Yes
No
End
Calculate fitness function for all wolves
Select X_α , X_β, X_δ Calculate patameter a
i=1
$i <=$ pop_size
YES
NO
$it=it+1$
Calculate parameters r_1, r_2, A, C
Determine the leader
$|A| <1$
YES
NO
Converge X_i to the leader
Diverge X_i from the leader
$i=i+1$
Update S_i

Figure 2

Flowchart of SetGWO

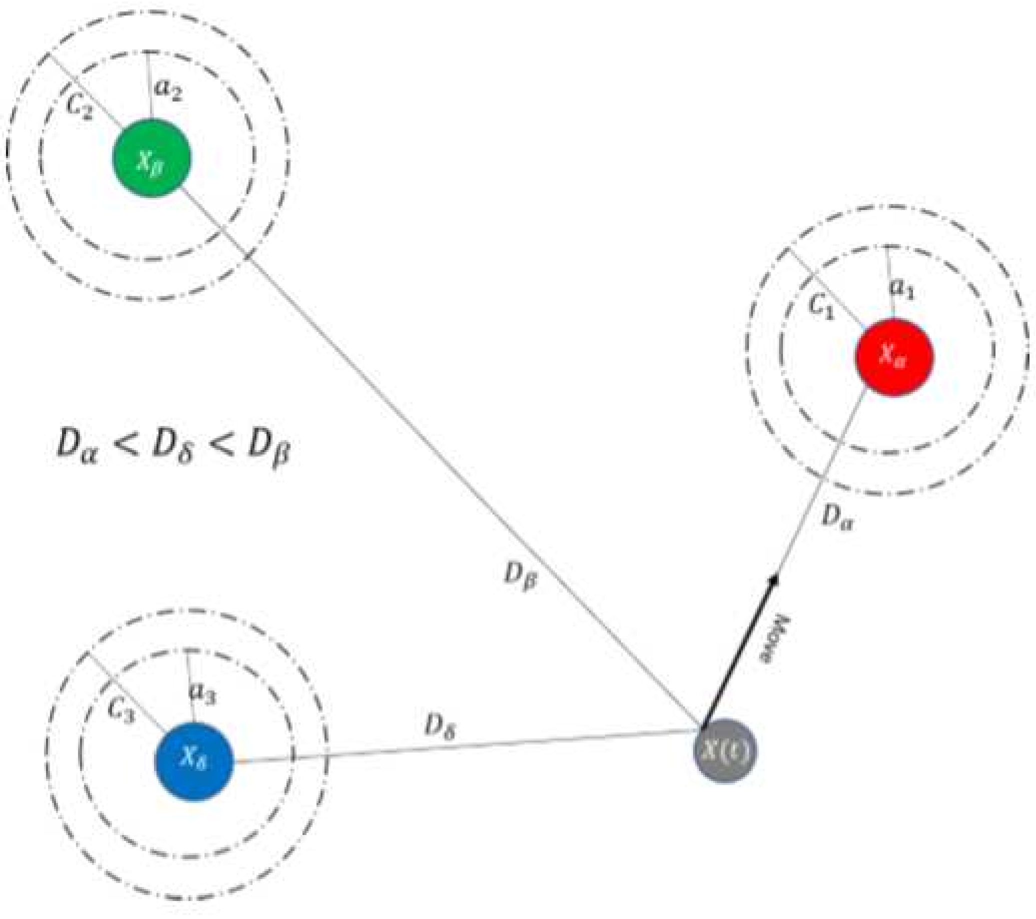


Figure 3

Encircling strategy in SetGWO

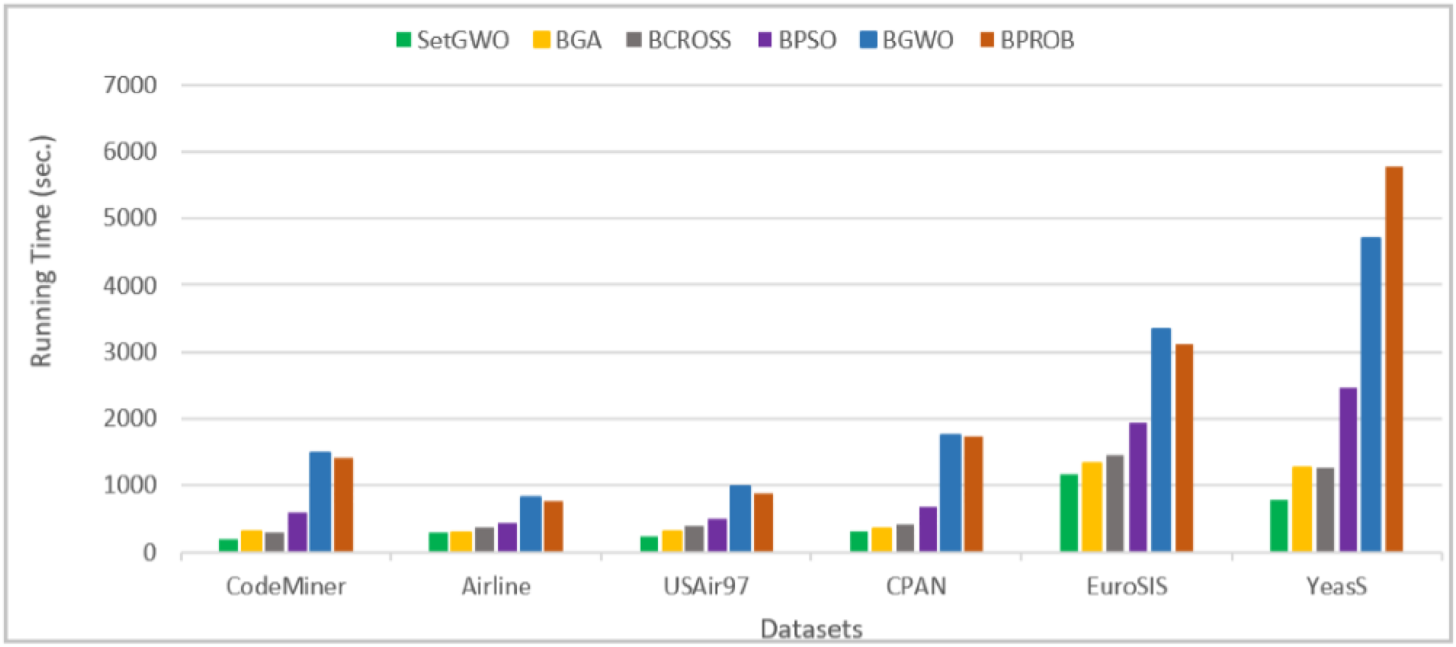


Figure 4

Running time comparison results on Vertex Cover

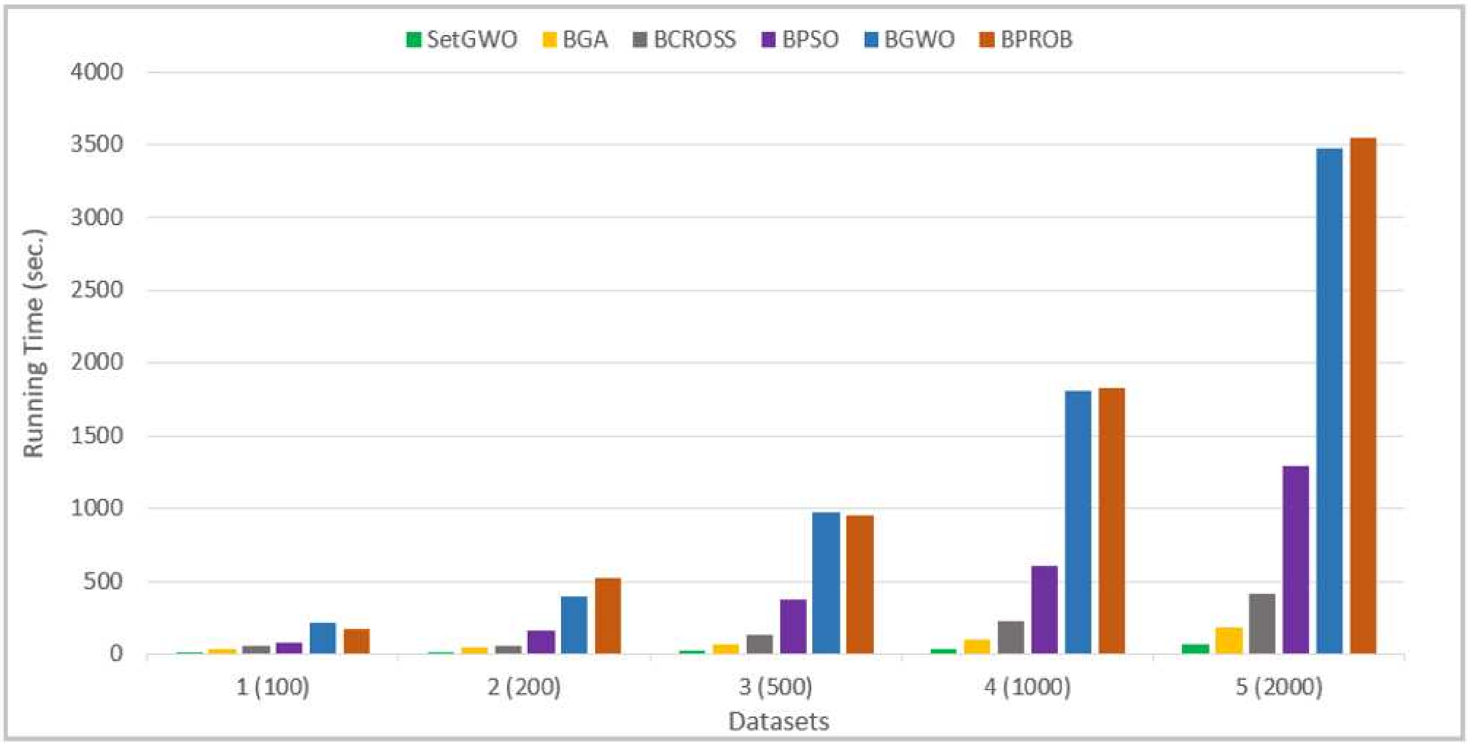


Figure 5

Running time comparison results on Knapsack

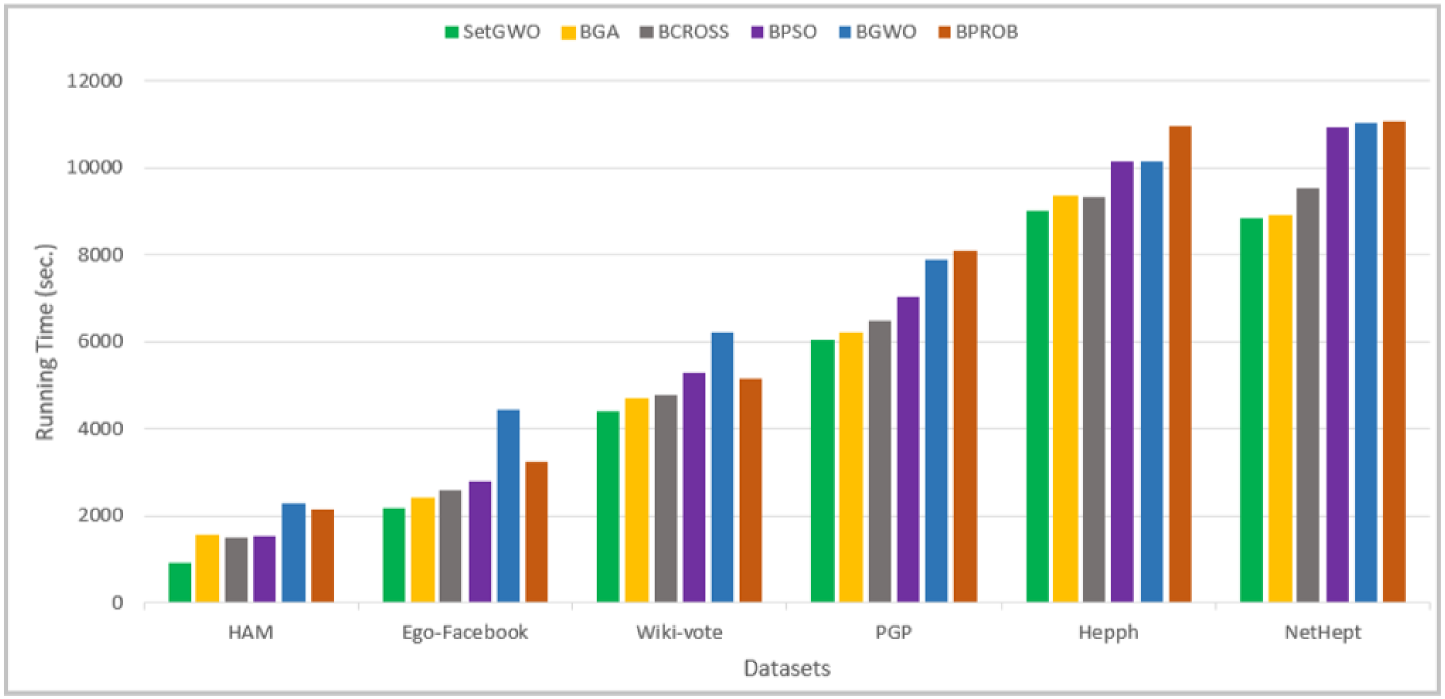


Figure 6

Running time comparison results on Influence Maximization

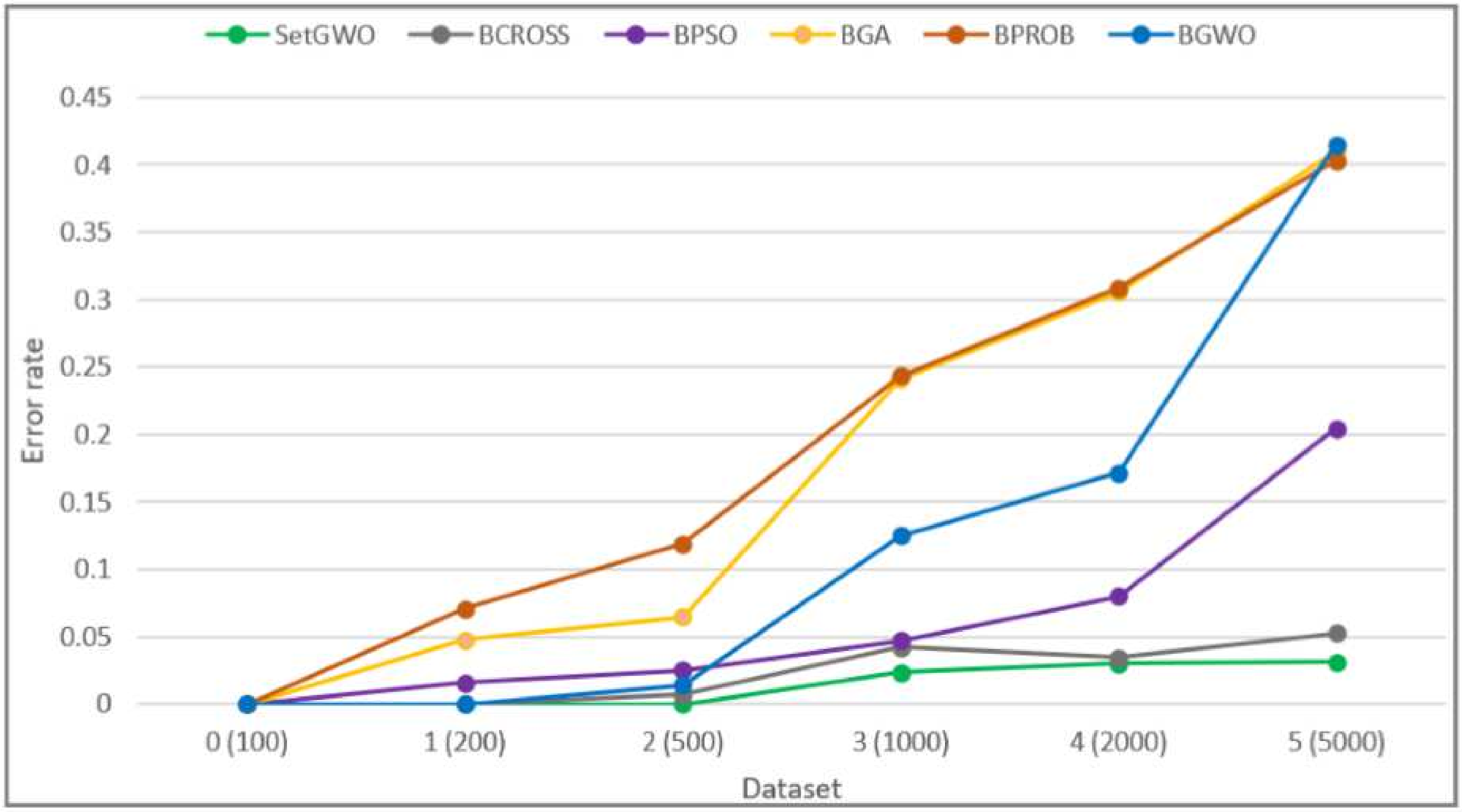


Figure 7

Scalability comparison results on Knapsack in terms of quality of solutions

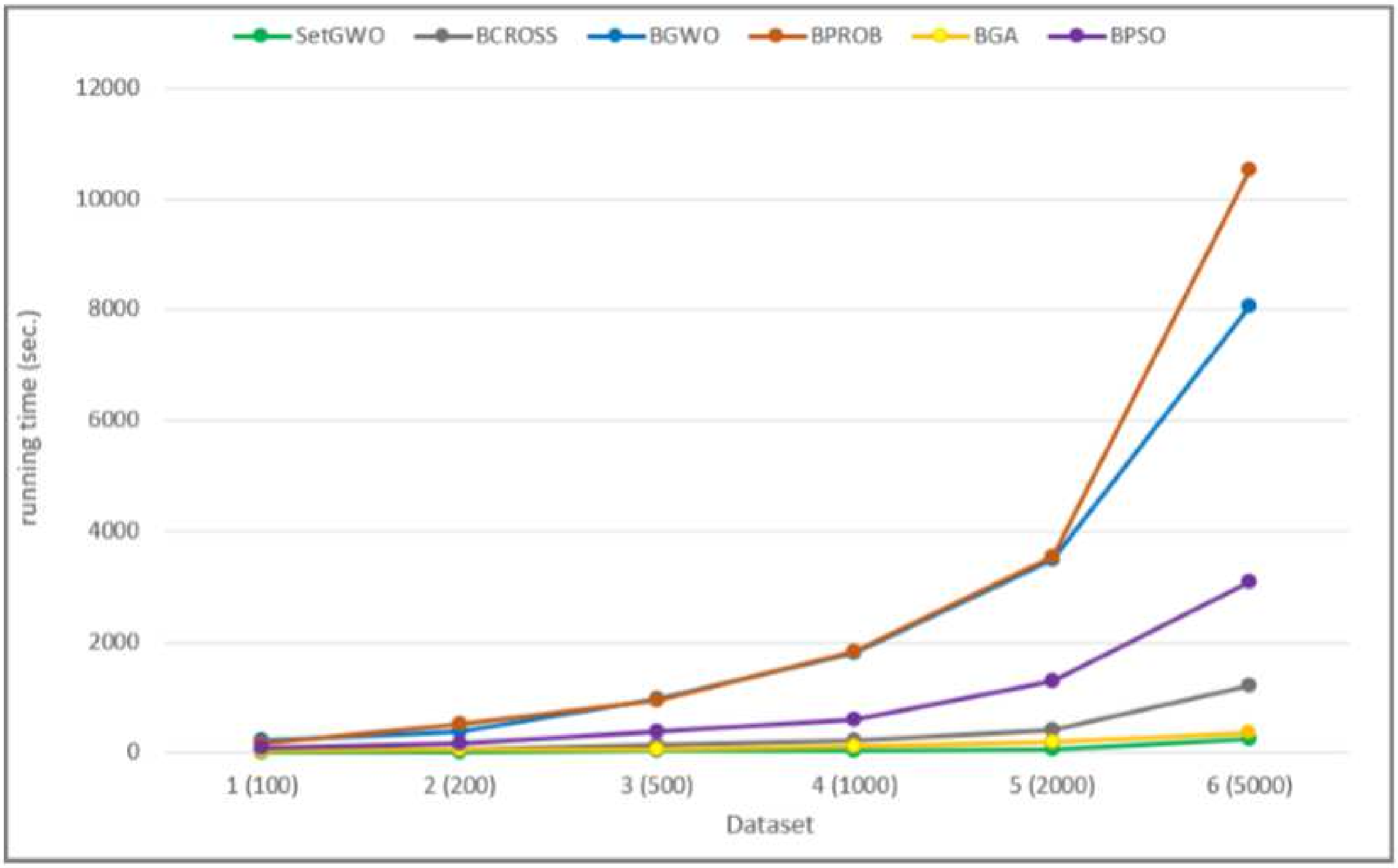


Figure 8

Scalability comparison results on Knapsack in terms of running time