An Improved Gray Wolf Optimization Algorithm for Solving Disassembly Sequencing Problems

Laide Guo1,a, Fangjie Peng2,b, Dapeng Song3,c, Chengbo Hu4,d and Ling Dou5,e

1The Computer and Communication Engineering College, Liaoning Petrochemical University, Fushun, 113001, P R. China.
2The Department of Biochemistry Engineering at Chaoyang Teachers College, Chaoyang, 122000, P R. China.
3The Police Command Intelligence Detachment, Public Security Bureau of Liaoning province, Chaoyang, 122000, P R. China.
4The Department of Education of Liaoning province, Shenyang, 110032, P R. China.
5Library of LNPU, Liaoning Petrochemical University, Fushun, 113001, P R. China.

a guolaide@lnpu.edu.cn; b qiliang@sdust.edu.cn; c 6544188@qq.com; d 9536285@qq.com; e dl_lnpu@163.com

Abstract. In recent years, the utilization of end-of-life products which may cause serious environmental pollution has received much attention from both industrial and academia. The recycling and reusing of waste products have an essential step that is disassembly. A disassembly sequencing planning problem is studied with minimizing disassembly time to get the near-optimal solution. This work uses an improved algorithm for solving disassembly sequencing problems based on an improved gray wolf optimization algorithm, which has a group optimization options that can simulate the gray wolves’ predation behaviours. An initial solution generator, a new solution generator, and a random mutation operator are adopted to improve the proposed algorithm, which can promote its efficiency and effectiveness. Experimental data show its superiority to solve this work’s problem comparing with the classical genetic algorithm.

1. Introduction
The rapid industrialization increases the technological waste, which has severe environmental impact. Governments in most countries have given manufacturers the responsibility to reuse the End-of-Life (EOL) products for environment friendly disposal [1]. Disassembly needs to balance the relationship between resources consumption and value generation [2, 3]. Disassembly sequence planning problem (DSP) means that it gets a group of optimal disassembly sequences by optimizing some objectives such as disassembly profit and time.

Most of researchers focus on deterministic disassembly sequence planning through heuristics or metaheuristics [4]. Guo et al. [5] optimize disassembly sequences according to the multi-resource constraints to obtain the expected maximum profit. Torres et al. [6] study the nondestructive automatic disassembly of personal computer. Smith et al. [7] Analysis of environmental cost and economic benefit in DSP. It focuses on finding an optimal stopping point in consideration of environmental cost and economic benefit. Kalaycı et al. [8] deal with uncertain task processing time in a disassembly problem. They apply triangular fuzzy membership functions to describe it and design an Artificial Bee Colony
(ABC) method to optimize the objective function. Ja et al. [9] propose an adaptive method to calculate the assigned value. Gray wolf optimizer (GWO) is designed to improve the effectiveness of the optimal solution. Xia et al. [10] take DSP as the core to minimize the disassembly cost. They put forward simplified teaching optimization algorithm to solve their proposed DSP.

Gray Wolf Optimizer (GWO) is proposed according to the simulation of gray wolf population hierarchy and predation behavior [11]. This project studies the possibility of applying an improved gray wolf optimizer (IGWO) to the DSP. It keeps the main idea of exploration and exploitation of GWO to solve these problems. To compare with the prior researches, we put forward three contributions:

1. It applies an improved gray wolf algorithm (IGWO) to solve the completed DSP.
2. It develops a new initial solution operator, a new mutation operator to deal with DSP and an initial random sequence generator.
3. It reduces a solution space of Discreteness and combination by IGWO, the performance of proposed algorithm is validated via experiments results according to two cases.

The rest of the paper is described as follows. The investigated problem is presented in Section 2, proposes and the proposed algorithm is described in Section 3. Section 4 presents simulation experiments on disassembly cases and discuss experimental data. Section 5 presents the conclusion and the future work.

2. Problem Statement

2.1 Notations

Some notations and variables are described as follows.

| Symbol | Description |
|--------|-------------|
| j      | Component index |
| s      | A disassembly sequence |
| t       | Disassembly time of component j in sequence s |
| d       | Penalty of direction change for disassembling component j in sequence s |
| m       | Penalty of method change for disassembling component j in sequence s |
| D       | Demand type of component j in sequence s |
| M       | Material type of component j in sequence s |
| T       | Disassembly time right after the disassembly operation of sequence s |
| T       | The total disassembly time |
| n       | The total number of parts in a product |
| s_i     | The i-th component of a product |
| a       | A variable |
| v       | A solution |
| P       | A population |
| r       | A random number |

2.2 Objective Function

The optimal objective is to solve the disassembly sequence within the least time. This product includes 13 components, each component includes a disassembly direction, the disassembly method, a type of demand and material. An precedence relationship of all the parts is shown in Figure 1 and Table 1. 13 components are represented by 1-13. Figure 1 shows the precedence relationships among components. For instance, components 1 and 2 must be disassembled before any others. We stipulate that +X, -X, +Y, -Y, +Z, -Z are used to disassemble components, respectively.

**Change in disassembly direction.** Six disassembly directions, i.e., +X, -X, +Y, -Y, +Z, -Z, can be used removing a component. $d_{ij}$ is the directional change penalty as (1)
\[ d_{js} = \begin{cases} 0, & \text{if no direction change required} \\ 1, & \text{if 90° change is required} \\ 2, & \text{if 180° change is required} \end{cases} \]

\[ m_{js} = \begin{cases} 0, & \text{no change required} \\ 1, & \text{change from N to D or D to N} \end{cases} \]

**Change in disassembly methods.** There are two kinds of methods to disassemble a component, i.e., destructively (D) disassembling or non-destructively (N) disassembling. \( m_{js} \) is the method change penalty as (2)

![Figure 1. Structure graph of a case](image)

**Table 1. Initial data for a case**

| \( j \) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| \( t_{i} \) | 4 | 1 | 2 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 3 |
| \( d_{js} \) | +x | +x | +x | -y | +z | -y | +z | -y | +x | -x | -y | +z |
| \( m_{js} \) | D | D | D | N | N | N | D | N | D | N | D | D |
| \( D_{js} \) | 1 | 1 | 0 | 0 | 1 | 2 | 1 | 2 | 0 | 1 | 2 | 1 |
| \( M_{js} \) | S | P | S | P | S | S | P | S | S | A | A | S | S |

**Change in component demand.** \( D_{js}=1 \) means that some components are demanded for reuse and \( D_{js}=2 \) means that some components are demanded for recycling. \( D_{js}=0 \) means that a component is not demanded.

**Change in component material.** The material of a component includes aluminum (A), plastic (P), or steel (S) in this product. Disassembly time is presented by \( t_{i} \), for a component, and \( T \) means the total disassembly operation time. The following is the cumulative time formula:

\[
T_{s} = \begin{cases} 
T_{s-1} + T_{i} + d_{ij} + m_{ij}, s = 0, \ldots, n-2 \\
T_{s-1} + T_{i}, s = n-1 
\end{cases}
\]

The algorithm 1 calculates time function. Since IGWO acquires the maximum value, we use a formula to change it, i.e., \( F = C - T \), where \( C > T \). In [1], \( C = 100 \).

**Algorithm 1 Calculate \( T_{s} \)**

| Input: \( D_{js}, M_{js}, t_{i}, d_{js}, m_{js} \) | (5) | \( T_{s} = T_{s-1} + t_{i} + d_{ij} + m_{ij} \) |
|---|---|---|
| Output: \( T \) | (6) | Else |
| (1) | For \( s = 0 \) to \( n-1 \) | (7) | \( T_{s} = T_{s-1} + t_{i} \) |
| (2) | If (\( D_{js} = D_{js+1} = 2 \) && \( M_{js} = M_{js+1} \)) then | (8) | End if |
| (3) | \( T_{s} = T_{s-1} \) | (9) | End for |
| (4) | Else if \( s \neq n-1 \) then | (10) | \( T = T_{s} \) |

2.3 **Fitness calculation cases**

Tables 2 and 3 show two cases of fitness evaluation. The disassembly sequence for Case 1 is \( 2 \rightarrow 1 \rightarrow 6 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 9 \rightarrow 10 \rightarrow 8 \rightarrow 12 \rightarrow 11 \rightarrow 13 \), as shown in Table 2, and Case 2 is \( 2 \rightarrow 1 \rightarrow 6 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 9 \rightarrow 10 \rightarrow 12 \rightarrow 8 \rightarrow 11 \rightarrow 13 \), as shown in Table 3. Note that, Case 2 gives a disassembly sequence in which the components are of the same material, e.g., \( s = 9 \) and 10. In Case 2, when \( s = 9 \) and 10, \( t_{10} = t_{12} = 2 \). Hence, in the first case, \( F = 100-39 = 61 \), but in the second case, \( F = 100-37 = 63 \).
3. Sequencing Gray Wolf Optimizer Algorithm

According to the distance between the prey and the gray wolf, the gray wolf is divided into the following four categories, i.e., α, β, δ, ω, which denote the first good solution, the second good solution, the third good solution, the fourth good solution, respectively. The α wolf is the leader of the whole group in the hunting process, and it is the most intelligent and has the best fitness and distance; β and δ wolves are the two individuals with the second good fitness. In hunting, they will assist the α wolf in managing and arresting the gray wolves. They are also the candidate of the α wolf. The remaining wolves are defined as ω, whose main responsibility is to balance the gray wolf population and assist α, β, δ attack the prey. In the hunting process, the α wolf is the first to track and approach hunting. The GWO algorithm can divide the hunting process into three stages, i.e., encirclement, hunting and attack [5], and finally capture the prey (get the optimal).

3.1 Initial Random Sequence Generator (IRSG)

In order to obtain near optimal solution using IGWO, Encoding is an important step to generate a feasible solution. A list of integers is used to represent each disassembly sequence, and one integer \( v \in \{1, 2, ..., n\} \) denotes a disassembled component.

A random number is generated according to the disassembly tasks and the number of components of a product. The random number represents the number of swaps needed to disassemble the product.

3.2 New Solution Generator (NSG)

In this section, we design a generator to get the new solution. If a new solution is superior to the solution in the initial population, it is added to a new population. If the number of individuals in a gray wolf population cannot meet the specified requirements, an extra child is generated by the δ wolf.

\[
N_i = na
\]

where \( N_i \) is the number of swaps. If we identify the first component, the second component is swapped via

\[
P_r = \frac{R}{2a}
\]

where a random number is used by \( R \in [0, 1] \), \( P_r \) denotes a component to be swapped when a component is selected in a feasible disassembly sequence.

3.3 Random Mutation Operator (RMO)

We design a random mutation operator to control the solution quality. The principle of this method to generate new solution is the same to a mutation operator of genetic algorithm [12]-[14]. It should be noted that (5) determines RMO. For each individual, after NSG operation, there is a 20% chance that RMO operation will change the solution. The solution random exchange finite times, but we can control

| Table 2. Fitness calculation of case 1 | Table 3. Objective function of case 2 |
|-----------------------------------|-----------------------------------|
| \( s \) | \( j \) | \( t_j \) | \( d_j \) | \( m_j \) | \( D_j \) | \( M_j \) | \( F \) | \( s \) | \( j \) | \( t_j \) | \( d_j \) | \( m_j \) | \( D_j \) | \( M_j \) | \( F \) |
| 1 | 2 | 1 | +x | D | 1 | P | 0+1+0+0=1 | 1 | 2 | 1 | +x | D | 1 | P | 0+1+0+0=1 |
| 2 | 1 | 4 | -x | D | 1 | S | 1+4+1+0=6 | 2 | 1 | 4 | -x | D | 1 | S | 1+4+1+0=6 |
| 3 | 6 | 2 | -y | N | 2 | S | 6+2+1+1=10 | 3 | 6 | 2 | -y | N | 2 | S | 6+2+1+1=10 |
| 4 | 3 | 1 | +x | D | 1 | S | 10+1+1+1=13 | 4 | 3 | 1 | +x | D | 1 | S | 10+1+1+1=13 |
| 5 | 4 | 2 | -y | N | 0 | P | 13+2+1+1=17 | 5 | 4 | 2 | -y | N | 0 | P | 13+2+1+1=17 |
| 6 | 5 | 2 | +z | N | 1 | P | 17+2+1+0=20 | 6 | 5 | 2 | +z | N | 1 | P | 17+2+1+0=20 |
| 7 | 7 | 1 | +x | N | 1 | A | 20+1+0+0=21 | 7 | 7 | 1 | +z | N | 1 | A | 20+1+0+0=21 |
| 8 | 9 | 1 | +x | N | 0 | A | 21+1+1+0=23 | 8 | 9 | 1 | +x | N | 0 | A | 21+1+1+0=23 |
| 9 | 10 | 2 | -y | D | 1 | S | 23+2+1+1=27 | 9 | 10 | 2 | -y | D | 1 | S | 23+2+1+1=27 |
| 10 | 8 | 2 | -y | D | 2 | A | 27+2+6+0=39 | 10 | 12 | 2 | -y | D | 1 | S | 23+2+1+1=27 |
| 11 | 12 | 2 | -y | D | 1 | S | 29+2+6+0=39 | 11 | 8 | 2 | -y | D | 2 | A | 27+2+6+0=39 |
| 12 | 11 | 1 | -x | N | 2 | A | 31+1+1+1=34 | 12 | 11 | 1 | -x | N | 2 | A | 29+1+1+1=32 |
| 13 | 13 | 3 | +z | D | 1 | S | 34+3+1+1=39 | 13 | 13 | 3 | +z | D | 1 | S | 32+3+1+1=37 |
the number of exchanges. Thus, the operation expands the search space, and the local optimal value can be avoided searching at a local optimum.

$$N^*_i = \frac{1}{\alpha} \quad \text{(if } P_m < 0.2\text{)}$$

where the mutation probability is $P_m$, $\alpha$ is the number of swaps from formula (6). Note that the wolves converge to a single disassembly sequence. If RMO occurs in as wolf, the solution will mutate into a new solution which is more significant. The method strategy can shuffle itself according to the number of swaps as agreed in (6).

4. Experimental Results and Analysis

The IGWO is used to solve disassembly sequence problem of two products with different scales. The product structure of the second example is more complex than the product structure of the first example and therefore the disassembly process is more complex. Additionally, to further test the performance of IGWO, some comparisons are given and analyzed between GA and the well-known TILBO.

4.1 Case 1

A case in [1] is used to test the effectiveness of IGWO. Using IGWO and GA, the experimental results are shown in Table 4. We can see that IGWO is better than GA. Note that 200 times are executed by both algorithms for obtaining the experimental results. Table 4 shows that IGWO can find a better solution than GA. Although IGWO can find more near-optimal solutions, IGWO takes a lot of running time, which is about twice as long as GA. The reason is that the solutions meet precedence relationship of the components. To generate feasible solutions, a PPX operator is adopted by GA[1], while IGWO uses simply operator to generate solutions until it generates a feasible solution.

| Algorithm | $f_{best}$ | $f_{worst}$ | $f$ | std | Time (ms) |
|-----------|------------|-------------|-----|-----|-----------|
| IGWO      | 71         | 70          | 70.58 | 0.496 | 218.1     |
| GA        | 71         | 65          | 67.54 | 2.248 | 82.8      |

4.2 Case 2

In addition, we solve a product including 13 components with IGWO and GA at the same time.

Furthermore, the population size is set as follows: 50, 100, 150, 200, 250, respectively. From Table 5, it shows that the fitness of GA is lower than fitness of IGWO when both run 10 times. It means that IGWO can always find better solutions. However, the running time of IGWO to find the optimal solution is twice longer than that of GA.

Table 6. Results comparison among two algorithms for case 2

| No | Pop | Method | $f_{best}$ | $f_{worst}$ | $f$ | std  | Time (ms) |
|----|-----|--------|------------|-------------|-----|------|-----------|
| 1  | 50  | IGWO   | 70         | 68          | 69.1 | 0.737 | 3194.0    |

Table 4. Experimental results of IGWO and GA

Table 5. Results comparison among two algorithms for case 1
What’s more, the population size is set as follows: 50, 100, 150, 200, 250, respectively. From Table 6, it shows that the objective function of GA is lower than IGWO that of when they all run 10 times, which means that IGWO can get a better solution than GA. However, the running time of IGWO finding the near optimal solution is longer than GA. The standard deviation of IGWO running ten times is still small. It means that the IGWO is more stable.

In addition, we compare IGWO with a Teacher Learner Based Algorithm (STLBO) [4]. A case is shown in [4]. STLBO and IGWO are compared by using five test cases:
Case 1: $P = 10$, $I_c = 100$; Case 2: $P = 20$, $I_c = 50$; Case 3: $P = 50$, $I_c = 20$;
Case 4: $P = 100$, $I_c = 10$; Case 5: $P = 10$, $I_c = 250$
where $P$ is the population size. In test cases 1 – 4, we use the same data as provided in [4]. Note that we test Case 5 by using our proposed algorithm.

| Case | method | $f_{best}$ | $f_{worst}$ | $f$ | std | Run Time(ms) |
|------|--------|-------------|-------------|-----|-----|--------------|
| 1    | IGWO   | 33          | 35          | 33.66 | 0.74 | 1127         |
|      | STLBO  | 33          | 36          | 33.72 | 0.60 | 7160         |
| 2    | IGWO   | 33          | 35          | 33.76 | 0.75 | 1201         |
|      | STLBO  | 33          | 35          | 33.59 | 0.69 | 9502         |
| 3    | IGWO   | 33          | 35          | 33.50 | 0.68 | 1098         |
|      | STLBO  | 33          | 35          | 33.51 | 0.54 | 14052        |
| 4    | IGWO   | 33          | 35          | 33.59 | 0.496 | 1078         |
|      | STLBO  | 33          | 35          | 33.53 | 2.248 | 20142        |
| 5    | IGWO   | 33          | 35          | 33.37 | 0.54 | 15901        |

In Table 7, IGWO outperforms STLBO where $I_c$ is high and $P$ is small. We can see that from case 5, it shows that IGWO performs better than STLBO in [4]. Suppose you increase $I_c$ and $P$. It achieves the near optimal solution by using IGWO and it is better than all cases of STLBO in [4] from the mean of total disassembly time. We found that STD is the second lowest one among all data in Case 5.

5. Conclusion
From this work, IGWO is used and applied to deal with DSP. Through several groups of comparative experiments, it is proved that IGWO can find a near optimal disassembly sequence. We also designed IRSG, NSG and RMO operators. They improve the performance of the IGWO. We use some cases to check the IGWO algorithm. Experimental results know that the IGWO is stable and better than GA. Finally, we get satisfactory result, which shows that the IGWO has a good effect in solving DSP.

In the future, we plan to apply PPX and PMX (Partially Mapped Crossover) to IGWO, so we can find a new IGWO, whose fitness and time may be both better than GA. At the same time, we will also find some new algorithms have a good deal in DSP [15-24].

References
[1] E. Bazan, M. Y. Jaber, and A. M. A. El Saadany, “Carbon emissions and energy effects on manufacturing–remanufacturing inventory models,” Comput. Ind. Eng., vol. 88, pp. 307-316, 2015.
[2] M. Mandolini, C. Favi, M. Germani and M. Marconi, “Time-based disassembly method: how to assess the best disassembly sequence and time of target components in complex products,” Int. J. Adv. Manuf. Tech., vol. 95, no. 1-4, pp. 409-430, 2018.

[3] M. Marconi, M. Germani, M. Mandolini and C. Favi, “Applying data mining technique to disassembly sequence planning: a method to assess effective disassembly time of industrial products,” Int. J. Prod. Res., 2018, 10.1080/00207543.2018.1472404.

[4] K. Xia, L. Gao, W. Li, and K.-M. Chao, “Disassembly Sequence Planning Using a Simplified Teaching-Learning-Based Optimization Algorithm,” Sustainable Manufacturing and Remanufacturing Management, pp. 319–343, 2018.

[5] X. W. Guo, S. X. Liu, M.C Zhou, and G. D Tian., "Dual-Objective Program and Scatter Search for the Optimization of Disassembly Sequences Subject to Multiresource Constraints," IEEE Trans. Auto. Sci. Eng. vol. pp, no. 99, pp. 1-13, Aug 2017.

[6] Torres F, Gil P, Puareas J, Aracil R (2004) Automatic PC disassembly for component recovery. Int J AdvManuf Technol 23(1/2):39–46.

[7] S. Smith, L. Y. Hsu and G. C. Smith, “Partial disassembly sequence planning based on cost-benefit analysis,” J. Clean. Prod., vol. 139, pp. 729-739, 2016.

[8] C. B. Kalayci, A. Hancilar, A. Gungor, S. M. Gupta, “Multi-objective fuzzy disassembly line balancing using a hybrid discrete artificial bee colony algorithm,” J. Manuf. Syst., vol. 37, pp. 672-682, 2015.

[9] Luo Jia, Tang Bin. Application of new grey wolf optimization algorithm in function optimization[J]. Journal of Lanzhou University of Technology, 2016, 6(3): 97-101.

[10] K. Xia, L. Gao, W. D. Li and K. M. Chao, “Disassembly sequence planning using a simplified teaching–learning-based optimization algorithm,” Adv. Eng. Inform., vol. 28, no. 4, pp. 518-527, 2014.

[11] S. Mirjalili, S.M. Mirjalili A. Lewis "Grey wolf optimizer" Adv. Eng. Softw. vol. 69 no. 3 pp. 46-61 2014.

[12] X. W. Guo, M. C. Zhou, F. Alsokhiry, and K. Sedraoui, “Disassembly sequence planning: a survey,” IEEE/CAA J. Autom. Sinica, 2020. DOI: 10.1109/JAS.2020.1003515

[13] L. Qi, W. J. Luan, X. Y. Lu, X. W. Guo, “Shared P-Type logic petri net composition and property analysis: a vector computational method,” IEEE Access, 2020, doi:10.1109/ACCESS.2020.2974340.

[14] X. W. Guo, M. C. Zhou, S. X. Liu, L. Qi, “Multi-resource Constrained Selective Disassembly with Maximal Profit and Minimal Energy Consumption”, IEEE Trans. Autom. Sci. Eng., vol.18, no.2, pp. 804-816, April 2021.

[15] X. W. Guo, S. X. Liu, M. C. Zhou, G. D. Tian, “Disassembly sequence optimization for large-scale products with multiresource constraints using scatter search and Petri nets,” IEEE Trans. Cybern., vol. 46, no. 11, pp. 2435-2446, 2016.

[16] Y. Y. Tan, M. C. Zhou, Y. Y. Wang, X. W. Guo, L. Qi, “A hybrid MIP-CP approach to multi-stage scheduling problem in continuous casting and hot rolling processes, IEEE Trans. Autom. Sci. Eng., vol. 16, no. 4, pp. 1860-1869, 2018.

[17] X. W. Guo, M. C. Zhou, S. X. Liu, L. Qi, “Lexicographic Multiobjective Scatter Search for the Optimization of Sequence-Dependent Selective Disassembly Subject to Multiresource Constraints” IEEE Transactions on Cybernetics., vol.50, no.7, pp.3307-3317,July 2020.

[18] Z. Y. Zhao, S. X. Liu, M. C. Zhou, X.W. Guo, L. Qi, “Decomposition method for new single-machine scheduling problems from steel production systems,” IEEE Trans. Autom. Sci. Eng., vol. 17, no. 3, pp. 1376-1387, July 2020.

[19] Z. Zhao, S. Liu, M. Zhou, D. You, and X. Guo, "Heuristic Scheduling of Batch Production Processes Based on Petri Nets and Iterated Greedy Algorithms," in IEEE Transactions on Automation Science and Engineering, 2020. doi: 10.1109/TASE.2020.3027532.
[20] J. Zhao, S. X. Liu, M. C. Zhou, X. W. Guo, and L. Qi, “Modified cuckoo search algorithm to solve economic power dispatch optimization problems,” IEEE/CAA J. of Autom. Sinica, vol. 5, no. 4, pp. 794-806, July 2018.

[21] H. Liu, M. Zhou, X. Guo, Z. Zhang, B. Ning, and T. Tang, “Timetable optimization for regenerative energy utilization in subway systems,” IEEE Trans. on Intelligent Transportation Systems, accepted.

[22] Z. Y. Zhao, M. C. Zhou, and S. X. Liu, "Iterated Greedy Algorithms for Flow-shop Scheduling Problems: A Tutorial," in IEEE Transactions on Automation Science and Engineering, 2021. doi: 10.1109/TASE.2021.3062994.

[23] J. Zhao, S. Liu, M. Zhou, X. Guo and L. Qi, “An Improved Binary Cuckoo Search Algorithm for Solving Unit Commitment Problem–Methodological Description,” IEEE Access, vol.6, no.1, pp. 43535-43545, 2018.

[24] Z. Zhao, S. Liu, M. Zhou and A. Abusorrah, "Dual-objective mixed integer linear program and memetic algorithm for an industrial group scheduling problem," in IEEE/CAA Journal of Automatica Sinica, doi: 10.1109/JAS.2020.1003539.