# Applying Modified Colonial Competitive Algorithm to Solve Minimal Hitting Set Problems

Chuanjun Zhu[1], Jing Cao[1], Chaoyong Zhang[2], and Kunlei Lian[2]

[1]Hubei University of Technology, Wuhan, China, 430068

[2]State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan, China, 430074

October 6, 2019

**Abstract**

A CCA was developed and modified to solve the problem of minimal candidates set. The minimal candidate set was a minimal hitting set. The modified CCA improved performance in initialization, assimilation and rebirth of original CCA by introducing a third type of country, independent country, to the population of countries maintained by CCA by introducing a third type of country, independent country, to the population of countries maintained by CCA. Implementation details of the proposed CCA and modified colonial competitive algorithm (MCCA) were elaborated using an illustrative example. The performance of the algorithms was analyzed, and the results by the MCCA were compared with DMDSE-Tree algorithm. When 90% of the minimal hitting sets are obtained, the MCCA has better efficiency. Finally, the experimental results of certain system verify the effectiveness of the algorithm, which proves that this method can be applied in solving the minimal hitting set of combinatorial optimization problems for selection of equipment effectively.

***Key words:*** Business planning; colonial competitive algorithm (CCA); minimal hitting set; equipment selection

## 1 Introduction

Many real-world problems can be modeled as the minimal hitting set problem, which is defined as the set with minimum cardinality that has a non-empty intersection with each set in a collection of sets. Examples

include the creation of model-based diagnosis from the smallest conflict set, the search for gene fragments to expound certain gene characteristics in gene expression analysis and the decision problem when collective users make book purchases at public expense. There have been lots of efforts in the research community to improve algorithm efficiency in solving the minimal hitting set problem, which is NP-hard. Reiter (Reiter, 1987) proposed the HS-Tree method to identify the minimal hitting set in model-based diagnosis. It utilizes pruning strategy and closing strategy in the solution process and may encounter the problem of losing feasible solutions. To overcome this issue and find all the possible minimal hitting sets, Greiner (Greiner et al., 1989) proposed an improved HS-Tree algorithm named HS-DAG based on the idea of acyclic graphs. The minimal hitting set problem has received lots of attentions from the research community and many algorithms have been proposed to solve it more effectively, which can be classified into exact algorithms and heuristic algorithms. Examples of exact algorithms include the algorithm based on BNB-HSSE (Xiaomei Chen and Qiao), hybrid algorithm based on CHS tree and recursive boolean algorithm (Wang et al., 2010), hybrid algorithm based on HSSE tree and binary mark (Feng et al., 2011), branch reduction algorithm (Shi and Cai, 2010) and algorithm based on dynamic maximum degree (DMDSE-Tree) (Liming Zhang and Zeng, 2011). These exact methods cannot satisfy the time and space requirements of engineering problems in large scale complex systems, and, from a practical application perspective, it is often not necessary to find the optimal solutions in these systems. Therefore, heuristic algorithms have been proposed to approximate the optimal minimal hitting sets using algorithms like discrete particle swarm algorithm (Ronghua Jiang and Long, 2009), binary particle swarm optimization (Xiaoming Lv and Lian, 2012), and low-cost heuristic algorithm STACCATO based on heuristic functions (Abreu and Van Gemund).

Atashpaz-Gargari et al. (Atashpaz-Gargari and Lucas, 2007) proposed a population-based colonial competitive algorithm (CCA) by simulating the process of colonial competition in human society evolvement. It demonstrates better convergence rate when compared to genetic algorithm and particle swarm optimization, and has received more and more applications in recent years. For example, Forouharfard et al. (Forouharfard and Zandieh, 2010) used the colonial competitive algorithm to solve the logistic planning problem in transshipment warehouses. Kaveh et al. (Kaveh and Talatahari, 2010) applied the colonial competitive algorithm to the structural optimal design problem. Nazari et al. (Nazari-Shirkouhi et al., 2010) employed the colonial competitive algorithm for the product outsourcing problems in mix-model production. Sarayloo et al. (Sarayloo and Tavakkoli-Moghaddam, 2010) applied the colonial competitive algorithm in solving the dynamic unit manufacturing problem. In addition, numerous attempts have been made to improve the performance of colonial competitive algorithm and apply it to other combinatorial optimization problems (Hadji and Vahidi, 2011), including mixed-model assembly line balancing problem (Bagher et al., 2011) and the data clustering problem (Niknam et al., 2011). In this paper, we propose a modified colonial competitive

algorithm (MCCA) to solve the minimal hitting set problem and also validate its performance using the enterprise equipment selection problem.

# 2 Modified Colonial Competitive Algorithm for the Minimal Hitting Set Problem

In this paper, we propose a modified colonial competitive algorithm by introducing the concept of independent countries in addition to the existing two types of countries, namely, empires and colonies, in classical CCA. The three types of countries are defined and explained as follows:

- Empire: empires strive to assimilate and possess more colonies in the computational process of the algorithm.
- Colony: colonies aim to learn from their corresponding empires in order to grow stronger, with the objective of becoming either independent countries or new empire countries.
- Independent country: independent countries try to learn from existing empires and aim to become new empires.

With the introduction of independent countries, the MCCA differs from the classical CCA in a number of algorithm stages, including initial country construction, and assimilation and update within empires. The following sections elaborate these steps in details.

## 2.1 Empire initialization

Given the existence of three types of countries, MCCA uses three parameters to denote the number of each type of country in the initial population, namely, $N_{imp}$, $N_{col}$ and $N_{ind}$, which indicate the number of empires, colonies and independent countries, respectively. The total number of countries in the initial population is therefore $N_{pop} = N_{imp} + N_{col} + N_{ind}$. In the initialization step, the algorithm first randomly creates $N_{pop}$ solutions (countries) and selects the best $N_{imp}$ countries as empires, the $N_{col}$ colonies are selected next, and the remaining $N_{ind}$ countries are independent countries. The empire construction process in MCCA is the same with that of CCA. Note that independent countries are not influenced by any empire and therefore not part of any empires. Figure 1 depicts the country classification in the initialization stage.

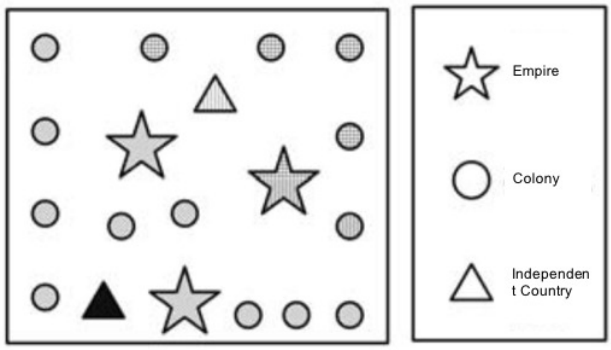



Figure 1: Empire initialization of MCCA

## 2.2 Solution encoding

Solution encoding scheme is the key step in applying CCA to any optimization problems and will greatly affect its performance thereafter. This paper uses the binary encoding theme, which is one of the most widely used encoding methods due to its simplicity for implementation.

For a set collection $U$, an assimilation step is first conducted to remove all the sets that encompass any other sets, which can greatly reduce the number of hitting sets. Only the encompassed sets, denoted by $U_i$, are left after the assimilation process. Let $n$ indicate the number of sets available in the set collection $U_i$ after the assimilation process. Let $R$ represent the union of all sets in the set collection, that is, $R = \cup_{i=1,2,\cdots,n} U_i, |R| = N$. It is clear that the minimal hitting set is a proper subset of $R$ and every element in $R$ either exists or does not exist in the minimal hitting set. Therefore, a binary value can be used to indicate whether an element in $R$ shows up in the minimal hitting set. In this way, the minimal hitting set can be represented by an array of length $N$ and each element in the array must be a binary value of either 0 or 1.

In order to create good approximations of the minimal hitting set in the initial population, reduce the number of algorithm iterations, and decrease the minimizing calculations, the average number of array elements with value of 1 is set to be close to the number of elements in the minimal hitting set. Let $X$ denote the average number of elements in the minimal hitting set and $\beta$ be the probability that an element takes the value of 1, then,

$$\beta = X/N \tag{1}$$

Since the value of $X$ is unknown in general, the value of $\beta$ must be approximated, which is empirically set to $0.1 \sim 0.5$. Note that the value of $\beta$ increases when the number of elements in the set, $n$, increases.

Let $Y$ be a chromosome and the gene at the $i$th ($i < N$) position be $x_i$, then $Y = \{x_1, x_2, \cdots, x_N\}$. The following method is used to create the initial solution based on the binary encoding scheme: randomly

create a number $\xi_i \in (0,1)$, let $x_i = 1$ if $\xi_i < \beta$; let $x_i = 0$ if $\xi_i \geq \beta$.

### 2.3 Fitness function

The definition of fitness function plays a key role in an algorithm's search efficiency. The fitness value shows the competitiveness of an solution regarding the objective function and it is also used in the algorithm to decide the solution's capacity to produce offsprings. Fitness function is often called objective function and is normally used to evaluate individual solution's quality. The objective function differs for different problems.

In order to assign an objective value to an individual, its closeness to the optimal solution must be evaluated. For the minimal hitting setting problem, there are two necessary conditions: 1) a solution must be a hitting set; 2) any of the solution's proper subset must not be a hitting set. This paper uses a objective function defined as $f(x) = t$, where $t$ indicates the number of non-empty intersections of the current solution with every set in the set collection. Note that the number of elements in a set cannot decide whether any proper subset of a set is a hitting set. The objective function drives a solution to converge to a hitting set, not necessarily a minimal hitting set, and the transformation of the resulting hitting set into a minimal hitting set is required.

### 2.4 Minimizing operator

As discussed in the previous section, the hitting set produced by a solution is not necessarily a minimal hitting set, and the purpose of the minimizing operator in this paper is to convert the resulting superset into a corresponding minimal hitting set. To this purpose, all individual gene from an solution with value of 1 are converted into 0 and the fitness value is computed. If the fitness value does not change, the conversion is saved. This conversion is repeated for each gene within a solution until all genes are converted and the individual will produce a minimal hitting set.

Within an individual, the genes with values of 1 are often related, meaning that whether we can convert a gene from 1 to 0 is affected by other genes with values of 1. To decide whether elements are related, we need to check the locations corresponding to the elements after assimilation. If there are two or more elements in the assimilated set, the elements are related; otherwise, they are not related.

If we scan all the genes within a solution chromosome sequentially starting from the first gene, we can find that the probability of converting genes from 1 to 0 is higher for those located in the front of the chromosome than those in the back. In this case, individuals represented by genes in the front part of the chromosome have higher probability to show up in the minimal hitting set than those located in the back. In order to assign equal probability to genes regardless of their relative positions on the chromosome, a random scanning scheme is adopted to convert the superset into a minimal hitting set.

The minimizing operator works as follows:

- Check whether the current object is a hitting set or not, if so, go to step (2); otherwise make no changes to the object.
- Start scanning the chromosome at position $k$ where $k$ is a random number and $k < N$, convert a gene value from 1 to 0 if applicable and compute its objective value. Keep the change if the resulting solution is a hitting set; otherwise, reverse the change. Let $k = k + 1$. Let the scanning position be $k - N$ if $k \geq N$. After the whole chromosome is scanned, the resulting objective represents a minimal hitting set and copy the distinct components into the final solution.

## 2.5 Competition

Similar to the classical CCA, the MCCA needs to calculate the total energy of each empire, which is affected by all the energies of its colonies:

$$E_{C_n} = J_n + \alpha \times \text{mean}\{J_m\} \tag{2}$$

where $E_{C_n}$ is the total energy of the $n$th empire; $\alpha \in (0, 1)$ is used to control the weight of the all the colonies' energies in the final total energy of the empire. Note that a bigger $\alpha$ value indicates a smaller weight for the empire itself, whereas a smaller $\alpha$ value indicates a bigger weight for the empire. Normally, $\alpha$ is set to 0.1 In addition, $J_n$ represents all the colonies' energy and $J_m$ is the energy of the $m$th colony of the $n$th empire.

Another similarity between classical CCA and the MCCA is that all the empires try to possess more colonies to increase their total energies. During the algorithmic process, some empires will have more and more colonies, which will result in some other empires losing their colonies gradually. The colony competition process works as follows: 1) identify the weakest colony from the empire with the smallest total energy and set the colony as a status of free; 2) all the empires compete for the free colony. The competition result is decided using a probability value that is associated with the empire's strength, in other words, more powerful empire has higher probability of getting the free colony.

The probability computation requires the normalized total energy of an empire $E_{CN_n}$:

$$E_{CN_n} = E_{C_n} - \max_{n=1}^{N_{imp}}\{E_{C_n}\} \tag{3}$$

where $E_{C_n}$ and $E_{CN_n}$ represent the empire's total energy and normalized total energy, respectively. The

possession probability can therefore be computed as $P_n$:

$$P_n = |E_{CN_n} / \max_{n=1}^{N_{imp}} \{E_{C_n}\}| \tag{4}$$

In order to assign the free colony to an empire, construct the following vector:

$$P = (p_1, p_2, \cdots, p_{N_{imp}}) \tag{5}$$

and another evenly distributed vector $R$ of the same size:

$$R = (r_1, r_2, \cdot, r_{N_{imp}}), \quad r_1, r_2, \cdots, r_{N_{imp}} \in U(0,1) \tag{6}$$

then we get the following vector $D$:

$$D = P - R = (D_1, D_2, \cdots, D_{N_{imp}}) = (p_1 - r_1, p_2 - r_2, \cdots, p_{N_{imp}} - r_{N_{imp}}) \tag{7}$$

Based on the above computations, the free colony will be assigned to the empire with the biggest $D$ value. The competition process is shown in figure 2.

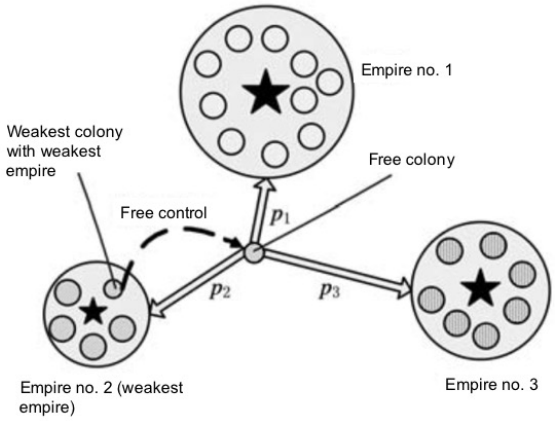



Figure 2: Illustration of colony competition in MCCA

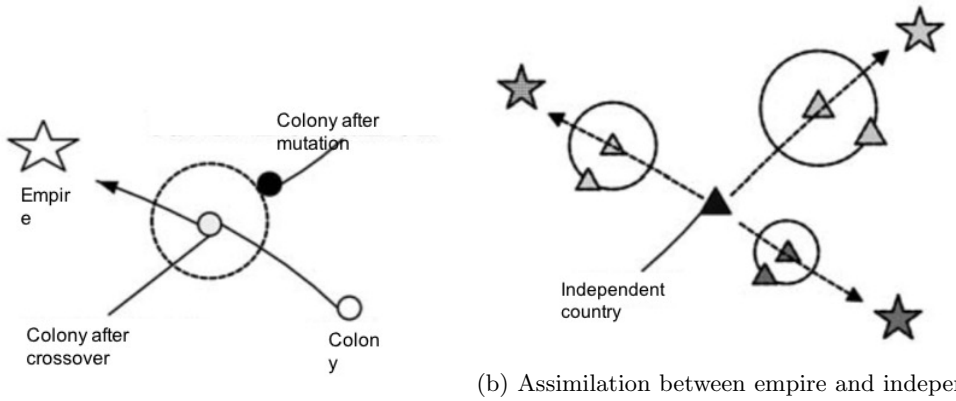
## 2.6 Assimilation

There exist two types of assimilation in the MCCA, namely, assimilation within an empire and assimilation between an empire and an independent country. The former assimilation process involves only the empire and its colonies and is the same with that of the classical CCA. The latter assimilation process is described

as follows:

- An independent country will tentatively move to all empires and the movement process is the same with that of colony to its empire. The movement process is achieved by the crossover and mutation operators. This paper uses the single-point crossover operator and the mutation operator randomly selects a gene to reverse its current value in order to generate a new solution.
- Separate new independent country is generated after the movement toward every empire and the best one is used as the final new independent country.

The assimilation processes are depicted in figure 3. It can be seen from the figure that empire assimilation only involves the empire and its associated colonies, which is different from the assimilation between an independent country and an empire. The attempts to move to every empires made by an independent country can speed up the algorithm's convergence rate by only selecting the best move; on the other hand, more computational times are required.




(a) Assimilation within empire

(b) Assimilation between empire and independent country

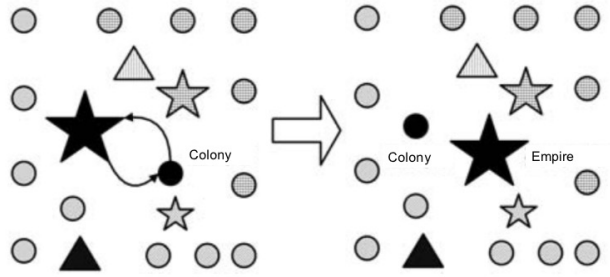
Figure 3: Illustration of assimilation of MCCA

## 2.7 Update

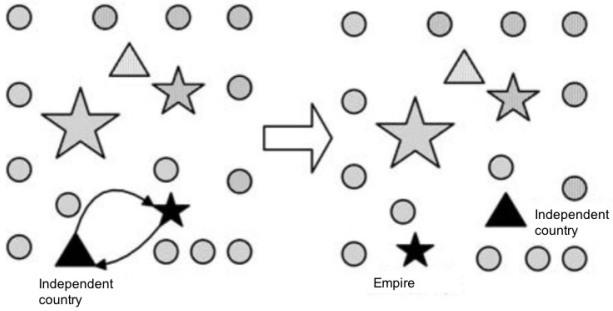
There are three types of updating processes in the MCCA, which is depicted in figure 4:

- Updating process between an empire and its colonies. If the new colony created after the assimilation process has better objective value than that of the empire, it will become the new empire. The process is illustrated in figure 4a.
- Updating process between an independent country and an empire. If the best independent country has better objective value than that of the weakest empire, it will replace the empire and the replaced empire will become an independent country. The process is depicted in figure 4b.

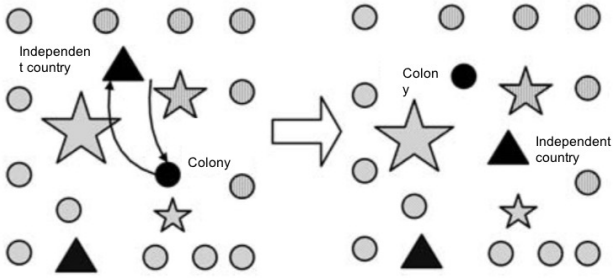
- Updating process between colonies and independent countries. If the best colony after assimilation has better objective value than that of the weakest independent country, it will replace the independent country, which will become a colony. The process is shown in figure 4c.




(a) Update between empire and colony




(b) Update between empire and independent country




(c) Update between colony and independent country

Figure 4: Illustration of updating processes in MCCA

### 2.8 Empire removal

The MCCA uses the same empire removal step as in the classical CCA.

## 3 Experiments

In this paper, we randomly generate 50 arrays of size 10 with values no greater than 20. All the arrays are sorted non-decreasingly and all the equal numbers are set to 0. The 50 arrays are then put into five

groups: 1) the first group encompasses arrays $1 \sim 10$; 2) the second group encompasses arrays $1 \sim 20$; 3) the third group encompasses arrays $1 \sim 30$; 4) the first group encompasses arrays $1 \sim 40$; 5) the first group encompasses arrays $1 \sim 50$; These generated instances are solved independently using the MCCA and the minimal hitting set algorithm based on dynamic maximum degree (DMDSE-Tree) (Liming Zhang and Zeng, 2011). For each group, the minimal hitting set and average computation times from five runs are recorded to make comparisons.

The experiments are run on a Windows XP computer with Intel Pentium G630 2.70 GHz CPU and 2GB memory. All the algorithms are implemented using Visual C++ 6.0. The parameters of MCCA are set as follows: $N_{pop} = 100$, $N_{imp} = 7$, $N_{ind} = 5$, the maximum number of iterations $N_{max} = 100$, the weight is set as $0.2, 0.3, \cdots, 0.8$ and $\alpha = 0.8$. The minimal hitting set and computational time comparisons between MCCA and DMDSE-Tree are given table 1 and table 2, respectively.

Table 1: Computational results using DMDSE-Tree

| group | size | No. MHS | computation time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | average time |
| 1 | 10 | 348 | 6.468 | 7.297 | 7.531 | 8.031 | 6.812 | 7.2287 |
| 2 | 20 | 861 | 21.932 | 21.671 | 22.734 | 20.750 | 22.172 | 21.8514 |
| 3 | 30 | 2426 | 43.343 | 43.109 | 43.297 | 43.688 | 44.250 | 43.5374 |
| 4 | 40 | 1618 | 61.516 | 59.562 | 59.953 | 58.828 | 58.921 | 59.7560 |
| 5 | 50 | 3165 | 89.797 | 88.406 | 88.562 | 87.281 | 87.187 | 88.2466 |

Table 2: Computational results using modified CCA

| group | size | No. MHS | computation time (s) | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | average time |
| 1 | 10 | 322 | 4.312 | 4.297 | 4.625 | 4.469 | 5.672 | 4.6570 |
| 2 | 20 | 766 | 15.219 | 16.110 | 16.078 | 16.156 | 16.032 | 15.9190 |
| 3 | 30 | 1489 | 29.656 | 29.875 | 31.234 | 30.453 | 30.906 | 30.4248 |
| 4 | 40 | 2017 | 43.140 | 43.078 | 42.907 | 43.938 | 43.826 | 43.3778 |
| 5 | 50 | 2909 | 62.656 | 66.219 | 63.016 | 64.762 | 62.318 | 63.9942 |

The experiments are run five times and the average computation times and minimal hitting sets are obtained. Table 3 shows the comparisons between computation time percentage and number of minimal hitting sets.

It can be seen from table 3 that MCCA is able to solve 90% of instances with about 70% of computational times when compared to DMDSE-Tree, which shows the superior performance of MCCA in solving minimal hitting set problems. In addition, MCCA shows better robustness in solving problems with various scales.

Table 3: Comparison results

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| MCCA time (s) | 4.6750 | 15.9170 | 30.4248 | 43.3778 | 63.9942 |
| DMDSE-Tree time (s) | 7.2278 | 21.8514 | 43.5374 | 59.7560 | 88.2466 |
| time percentage (%) | 64.48 | 72.84 | 69.88 | 72.59 | 72.52 |
| MCCA no. MHS | 322 | 766 | 1489 | 2017 | 2909 |
| DMDSE-Tree no.MHS | 348 | 861 | 1618 | 2426 | 3165 |

# 4 Application in Equipment Selection

In this section, we report an application of using the MCCA to solve the minimal hitting set problem encountered in a manufacturing company. There are 12 manufacturing units in the module-based production system and are denoted as $A, B, \cdots, L$, respectively. Every manufacturing unit consists of multiple general-purpose machines. The studied company plans to manufacture a new product $P1$, which requires 23 self-made parts. Multiple parts can be produced within the same units due to the small volume and extra processing capacity. The problem is to use minimal number of units to produce the new product in order to reduce the impact on existing production plan. Table 4 shows all the required parts and their corresponding units.

Table 4: Manufacturing units of parts

| part | manufacturing unit | part | manufacturing unit | part | manufacturing unit |
|---|---|---|---|---|---|
| 1 | A B F H | 9 | D H I J | 17 | A J K |
| 2 | C G H | 10 | E H J L | 18 | E L |
| 3 | D J L | 11 | H K | 19 | D H I |
| 4 | B I | 12 | D F L | 20 | B E J K |
| 5 | A B E G K | 13 | B G J | 21 | J L |
| 6 | C D H L | 14 | C J K L | 22 | G |
| 7 | D G H | 15 | E H K | 23 | E G K |
| 8 | A C J L | 16 | A C F H J | | |

The steps of using MCCA to solve the minimal hitting set problem to get the minimal number of production units are as follows:

- Assign numbers to each manufacturing unit. In this case, 1 is assigned to A, 2 is assigned to B. Repeat this assignment until L is assigned 12. Create an input file by setting 5 as the number of manufacturing units corresponding to the parts, all other units are set to 0.

- Identify the minimal hitting sets using the implemented C++ code using parameters $N_{pop} = 100$, $N_{imp} = 7$, $N_{ind} = 5$, $N_{max} = 100$ and weight set as 0.6. Table 5 shows the computational results. The minimal hitting sets that have the minimum number of elements are: $\{7, 8, 9, 11, 12\}$, $\{1, 7, 9, 11, 12\}$, $\{1, 2, 7, 8, 12\}$, $\{2, 7, 8, 10, 12\}$, $\{2, 7, 8, 11, 12\}$, $\{6, 7, 9, 11, 12\}$, $\{7, 8, 9, 10, 12\}$.

Table 5: Computational results and analysis

| MHS | no. elements | | MHS | no. elements |
|---|---|---|---|---|
| {2, 4, 5, 7, 8, 10 } | 6 | | {6, 7, 9, 11, 12} | 5 |
| {7, 8, 9, 11, 12 } | 5 | | {7, 8, 9, 10, 12 } | 5 |
| {1, 7, 9, 11, 12 } | 5 | | {2, 4, 6, 7, 11, 12 } | 6 |
| {1, 2, 7, 8, 12 } | 5 | | {4, 5, 6, 7, 9, 10, 11 } | 7 |
| {1, 5, 7, 8, 9, 12 } | 6 | | {2, 4, 5, 7, 10, 11 } | 6 |
| {2, 3, 4, 7, 11, 12 } | 6 | | {2, 3, 7, 9, 11, 12 } | 6 |
| {1, 2, 4, 7, 11, 12 } | 6 | | {2, 4, 7, 10, 11, 12 } | 6 |
| {2, 7, 8, 10, 12 } | 5 | | { 5, 6, 7, 8, 9, 10 } | 6 |
| {2, 7, 8, 11, 12 } | 5 | | {2, 7, 9, 10, 11, 12 } | 6 |
| {2, 5, 6, 7, 8, 10 } | 5 | | {3, 5, 6, 7, 9, 10, 11 } | 7 |
| {4, 5, 7, 8, 9, 10 } | 6 | | {1, 4, 5, 7, 9, 10, 11 } | 7 |

- Convert the hitting set with the minimal number of elements into manufacturing units: $\{G, H, I, K, L\}$, $\{A, G, I, K, L\}$, $\{A, B, G, H, L\}$, $\{B, G, H, J, L\}$, $\{B, G, H, K, L\}$, $\{F, G, I, K, L\}$, $\{G, H, I, J, L\}$.

Using the obtained manufacturing units to produce product $P1$, there is no need to change the production plans for other manufacturing units. It will incur less adjustment fees when product volume changes in the future.

# 5 Conclusions

The minimal hitting set problem has many real-world engineering applications but is computationally challenging to solve. This paper proposes an modified colonial competitive algorithm to solve this combinatorial optimization problem and elaborates the detailed algorithmic steps. Performance of the proposed algorithm is validated on testing instances and minimal hitting sets are obtained for most instances with reasonable times. Extensive comparisons are made during the computation process in order to get distinct minimal hitting sets, which slows down the solution process. Finally, the proposed algorithm is used to solve the minimal hitting set problem of equipment selection in a manufacturing system to further demonstrate its effectiveness.

# References

Rui Abreu and Arjan JC Van Gemund. A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis. In *Eighth Symposium on Abstraction, Reformulation, and Approximation*, pages 2–8.

Esmaeil Atashpaz-Gargari and Caro Lucas. Imperialist competitive algorithm: an algorithm for optimization

inspired by imperialistic competition. In *2007 IEEE congress on evolutionary computation*, pages 4661–4667. IEEE, 2007.

M Bagher, M Zandieh, and H Farsijani. Balancing of stochastic u-type assembly lines: an imperialist competitive algorithm. *The International Journal of Advanced Manufacturing Technology*, 54(1-4):271–285, 2011.

Wenquan Feng, Min Du, Qi Zhao, and Dong Wang. A method of combining hsse-tree and binary label to compute all minimal hitting sets. In *2011 Fourth International Symposium on Computational Intelligence and Design*, volume 2, pages 23–26. IEEE, 2011.

S Forouharfard and M Zandieh. An imperialist competitive algorithm to schedule of receiving and shipping trucks in cross-docking systems. *The International Journal of Advanced Manufacturing Technology*, 51 (9-12):1179–1193, 2010.

Russell Greiner, Barbara A Smith, and Ralph W Wilkerson. A correction to the algorithm in reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.

Moosa Moghimi Hadji and Behrooz Vahidi. A solution to the unit commitment problem using imperialistic competition algorithm. *IEEE Transactions on Power Systems*, 27(1):117–124, 2011.

A Kaveh and S Talatahari. Optimum design of skeletal structures using imperialist competitive algorithm. *Computers & structures*, 88(21-22):1220–1229, 2010.

Dantong Ouyang Liming Zhang and Hailin Zeng. Computing the minimal hitting sets based on dynamic maximum degree. *Journal of Computer Research and Development*, 48(2):209–215, 2011.

S Nazari-Shirkouhi, H Eivazy, Reza Ghodsi, Kamran Rezaie, and Esmaeil Atashpaz-Gargari. Solving the integrated product mix-outsourcing problem using the imperialist competitive algorithm. *Expert Systems with Applications*, 37(12):7615–7626, 2010.

Taher Niknam, Elahe Taherian Fard, Narges Pourjafarian, and Alireza Rousta. An efficient hybrid algorithm based on modified imperialist competitive algorithm and k-means for data clustering. *Engineering Applications of Artificial Intelligence*, 24(2):306–317, 2011.

Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987.

Shulin Tian Ronghua Jiang and Bing Long. Minimal hitting sets algorithm of identifying masking false failure sets based on dpso. *Systems Engineering and Electronics*, 31(4):997–1001, 2009.

Fatemeh Sarayloo and Reza Tavakkoli-Moghaddam. Imperialistic competitive algorithm for solving a dynamic cell formation problem with production planning. In *International Conference on Intelligent Computing*, pages 266–276. Springer, 2010.

Lei Shi and Xuan Cai. An exact fast algorithm for minimum hitting set. In *2010 Third International Joint Conference on Computational Science and Optimization*, volume 1, pages 64–67. IEEE, 2010.

Ziling Wang, Aiqiang Xu, Dingguo Wang, and Li Kong. Research on a new combined algorithm for computing minimal hitting sets. In *2010 International Conference on Measuring Technology and Mechatronics Automation*, volume 3, pages 14–17. IEEE, 2010.

Xiaofeng Meng Xiaomei Chen and Renxiao Qiao. Method of computing all minimal hitting set based on bnb-hsse. *Chinese Journal of Scientific Instrument*, 31.

Kaoli Huang Xiaoming Lv and Guangyao Lian. Generation of minimal candidate set for multiple fault diagnosis based on binary particle swarm optimization. *Systems Engineering and Electronics*, 34(5):961–965, 2012.