IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Journal Européen des Systèmes Automatisés
Vol. 54, No. 1, February, 2021, pp. 35-44
Journal homepage: http://iieta.org/journals/jesa

Discretization of Emperor Penguins Colony Algorithms with Application to Modular Product Design

Hayam G. Wahdan[1*], Hisham E. Abdelslam[1], Tarek H.M. Abou-El-Enien[1], Sally S. Kassem[1,2]

[1] Faculty of computers and Artificial Intelligence, Cairo University, Giza 12613, Egypt
[2] Smart Engineering Systems Research Centre, Nile University, Giza 12588, Egypt

Corresponding Author Email: Hayam@fci-cu.edu.eg

https://doi.org/10.18280/jesa.540105

Received: 1 September 2020
Accepted: 26 January 2021

Keywords:
Design Structure Matrix, Emperor Penguins Colony, Cuckoo Search, modular design, clustering

ABSTRACT

Modularity concepts attracted the attention of many researchers as it plays an important role in product design problems. Modularity requires dividing a product into a set of modules that are independent between each other and dependent within. The product is represented using Design Structure Matrix (DSM). DSM works as a system representation tool; it visualizes the interrelationship between product elements. In this research, a comparison is conducted between four optimization algorithms: Emperor Penguins Colony (EPC), First Modified Emperor Penguins Colony (MEPC1), Second Modified Emperor Penguins Colony (MEPC2) and Cuckoo Search (CS) optimization algorithms. These four algorithms aim at finding the optimal number of clusters and the optimal assignment of components to clusters, with the objective of minimizing the total coordination cost. Experimental results show that EPC outperforms the other three algorithms.

1. INTRODUCTION

Modular design has a significant impact on product development, which respond to market trends that require large varieties in small production processes [1]. Modularity is an important method that helps to manage large systems by breaking them into modules. Such modules should be interdependent within the same module and independent among different modules. Modular design requires clustering various components that form a product to create modules that are efficient and useful for production. Efficient modularity of products takes on greater significance as identical components are used in various products [2].

Design Structure Matrix (DSM) is a powerful tool which supports both analysis and complex system management [3]. It is also a product representation tool that allows the user to model, visualize and analyze the dependencies between the elements of a system. In addition, DSM helps to derive suggestions for the improvement or synthesis of a system, providing a clear visualization of the product design from the perspective of a product representation tool. A product can be represented by a DSM that contains a list of the components of the product. The DSM of commodity also offers exchange of knowledge and relationships of dependency between these components [4].

In this paper, four meta-heuristic algorithms are used to determine: (1) the optimal number of clusters in a DSM, and (2) the optimal assignment of components to clusters that minimizes the total coordination cost. These meta-heuristic algorithms are Emperor Penguins Colony (EPC), First Modified Emperor Penguins Colony (MEPC1), Second Modified Emperor Penguins Colony (MEPC2) and Cuckoo Search (CS) optimization algorithms. To the best of our knowledge, this is the first time EPC, MEPC1 and MEPC2 are used in solving discrete optimization problems.

The rest of this paper is structured as follows: Section 2 provides a brief introduction about DSM, followed by a review of related literature in Section 3. Problem definition and the used solution algorithms are provided in Sections 4 and 5, respectively. Section 6 includes numerical experimentation and analysis of the algorithms and, finally, the paper conclusion and future work are provided in section 7.

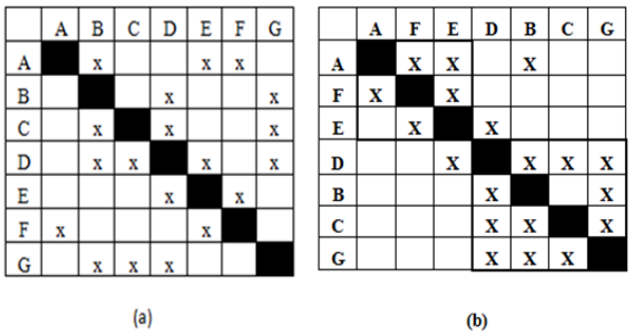
2. DESIGN STRUCTURE MATRIX (DSM)

The Design Structure Matrix (DSM) is a system analysis method that provides a complex system with a compact and consistent representation. It captures the interaction or interdependencies among elements of the system. DSM tolerates feedback and cyclic dependencies, an important feature as many engineering applications has the cyclic property [5].

DSM is a square matrix of size n, where n is the number of elements of the system. Figure 1 shows an example of a DSM of size 7. Elements names are written on the first raw and the first column of the matrix in the same order. An entry of 1 or x in the matrix means that the correspond elements i j (row_i, $column_j$) are dependent on each other. For each product, a DSM is developed and can be analyzed to classify modules, which are referred to as clustering. DSM clustering seeks to find clustering arrangements where modules interact with each other minimally and at the same time, components belonging to the same module interact maximally with each other [4].

For instance, Figure 1(a) Shows the original un-clustered DSM, Figure 1(b) shows a clustered DSM where the majority of the interactions are found in two domains, namely {A, F, E} and {D, B, C, G}. Figure 1(b) also shows that three

interactions do not belong to any specific module.



	A	B	C	D	E	F	G
A		x			x	x	
B				x			x
C		x		x			x
D		x	x		x		x
E				x		x	
F	x				x		
G		x	x	x			

(a)

	A	F	E	D	B	C	G
A		X	X		X		
F	X		X				
E		X		X			
D			X		X	X	X
B				X			X
C				X	X		X
G				X	X	X	

(b)

Figure 1. Example of a DSM of size 7 elements

3. LITERATURE REVIEW

Eppingeret, et al. [6] introduced the concept of minimizing module interactions, thus optimizing module interactions in a DSM. Idicula [7] suggested an algorithm for stochastic clustering of DSM clusters. Further developments were made by (Gutierrez 1998), where the development of a mathematical model. The model minimizes the total coordination cost [8].

A Genetic Algorithm is used to find optimal arrangement of elements within DSM which optimize the minimum description length (MDL) [9].

A new method is developed to define the difference between designing modular systems and integrative systems [10]. The study is focused on the specification of modules, modules architecture, and their interfaces. (DSM) is used and extended to represent more accurately the models under study [11].

To obtain better output from a clustering algorithm, a method known as conceptual module generation phase can be employed [12]. Liang [13] developed a model known as group decomposition model. The proposed model decomposes a complex set of activities into simpler ones. The DSM is used as a system simplification tool. The used clustering algorithm is K-means algorithm.

A modularization scheme based on functional modeling is proposed and K-means is used for clustering [14]. Neural networks algorithms and DSMs have been utilized to cluster DSM components with the objective function of clustering efficiency; however, the algorithm requires a predetermined number of clusters [15].

Borjesson and Hölttä [16] develop an algorithm named Idicula-Gutierrez-Thebeau Algorithm (IGTA) for clustering DSM. An improved algorithm, named IGTA-plus, is proposed. IGTA-plus provide significant improvement when compared with IGTA. Recorded improvements are in terms of computational time and solution quality. Genetic clustering is proposed with Minimum Description Length measure. A new assumption is added to minimize the total execution time. The proposed algorithm is tested on four case studies [17].

Yang et al. [18] developed a systematic clustering algorithm for organizational DSM. The algorithm evaluates clustering structures based on the strength of interaction Another novel approach for product design is introduced by integrating the sequence structure planning of assembly and disassembly of a product [19].

Clustering method is developed based on multidimensional scaling (MDS), this method is used DSM as system visualization tool [20]. Clustering algorithm using cuckoo search is developed to find optimal number of clusters within DSM and best assignment of each element in cluster using minimizing coordination cost as objective [4, 5]

A new practical method is proposed by Sakao et al. 2017 to support designers in creating service modules by extending the DSM [21]. novel research is developed, this research tries to answer a lot of questions, what is the importance of modularity concept in product design problems, how modularity helps in design for variety, and the importance of modulatiy in increasing the performance of the organization [22]. Multi-objective clustering algorithm is proposed using non-dominated cuckoo search to minimized coordination cost and maximizes Sustainability through DSM [23]. Finally, new research is developed aiming to construct independent clusters to be replaced or removed at any stage of manufacturing process [24].

The reviewed literature on product design using DSM as a system analysis tool revealed the existence of several techniques used to cluster the DSM for modularity. One major difference between those techniques is the objective of clustering. Minimizing cost is one of the most widely targeted objectives [25]. Another objective is the Minimal Description Length (MDL) [9], another clustering objective is the Clustering Efficiency (CE) index with static number of clusters [15].

From the solution technique point of view, several techniques were used in solving product design under modularity; stochastic hill-climbing algorithm [25], Genetic Algorithm [9], and neural networks [15]. To the best of our knowledge, this research is the first one to use the four previously mentioned metaheuristic algorithms with cost minimization as the objective function in solving product design problem under modularity while having the number of clusters dynamic.

4. PROBLEM DEFINITION

In modular design problem the product is represented using DSM, DSM is product representation tool, as shown in figure 1, DSM contains set of cells, cells contain “1” or “x” if there is relation between component i and j, cells contain “0” or “empty cell” if there isn’t relation between component i and j.

The objective is to cluster these components in such a way that minimizes the total coordination cost. Accordingly, two sets of decisions are to be considered; (1) the number of clusters to be formed, and (2) the optimal assignment of components in each cluster.

For a given DSM, the total coordination cost consists of two parts; IntraClusterCost and ExtraClusterCost as provided by Eqns. (1) and (2), respectively. If interaction DSM_{ik} belongs to cluster j then IntraClusterCost is to be calculated, otherwise ExtraClusterCost is to be calculated. The total cost is the additi0n of IntraClusterCost and ExtraClusterCost as shown in Eq. (3) and mentioned in the ref. [26].

$$intraClusterCost = \sum_{i,k \in Cluster\,j} (DSMik + DSMki) * \sum_{j=1}^{ncluster} (ClusterSize\,j)^{powcc} \quad (1)$$

$$ExtraClusterCost = \sum_{i,k \notin cluster\,j} (DSMik + DSMki) DSMSize^{powcc}, \quad j = 1 \ldots ncluster \tag{2}$$

where, DSM_{ik} is the relation between component i and k, DSMSize is the number of components (rows) in the matrix, powcc is a value utilized to penalize clusters' sizes, and ncluster is the total number of clusters. Clustersize(j) is the number of components within cluster j.

$$\text{Total coordination Cost} = \text{IntraClusterCost} + \text{ExtraClusterCost} \tag{3}$$

In this problem we have one constaint, this constaint is; each component must assign to one cluster; overlapping between clusters not allowed. overlapping between cluster decreases the importance of the clustering process and minimize the Sustainability of product being easy to replace or remove any cluster to produced different product or adding new features.

5. SOLUTION ALGORITHMS

Meta-heuristic optimization algorithms are general iterative algorithms capable of solving combinatorial optimization problems. These algorithms are stochastic in nature, and they simulate the behavior of particles. Meta-heuristic optimization algorithms try to find optimal or near optimal solutions for complex problems [27]. To solve the problem defined in Section 4, four meta-heuristic algorithms are utilized. These algorithms represent population-based optimization algorithms. The utilized algorithms are Emperor Penguins Colony (EPC), First Modified Emperor Penguins Colony (MEPC1), Second Modified Emperor Penguins Colony (MEPC2) and Cuckoo Search (CS) optimization algorithms. In the following subsections, a brief description of the algorithms is given, Section 5.5 shows the discretization process of EPC, MEPC1, and MEPC2.

5.1 Cuckoo Search (CS)

Yang and Deb [28] had proposed the Cuckoo Search (CS) algorithm. The algorithm simulates cuckoo birds' behavior to explore the solution space for an optimum solution, or near optimum. CS is inspired by the behavior of certain species of brood parasite cuckoo that lay their eggs in the nests of other host birds. Brood parasite cuckoos distribute their eggs between various nests. Their aim is to escape the parental investment in raising their offspring, and to minimize the risk of their egg loss, as mentioned by Yang and Deb [28].

One of the major advantages of CS is its performance, which has been proved by a large number of benchmark studies. CS performed better when comparing outcomes with other metaheuristic algorithms [29]. Another advantage its simplicity compared to other metaheuristic algorithms, because it involves setting only two parameters. This function simplifies the time and effort required to adjust and fine-tune the parameter settings for the algorithm.

5.2 Emperor Penguins Colony (EPC)

A new Meta heuristics algorithm named The Emperor Penguin Colony (EPC) was proposed by Harifi et al. [30]. The algorithm inspired by the behavior of emperor penguins in colonies when they move from a cold domain to a warmer one following a logarithmic spiral like movement. This algorithm is controlled by the body heat radiation of the penguins and their spiral-like movement in their colony. The algorithm tries to find optimal or near to optimal solution.

In the EPC, the temperature around the huddle is calculated, the algorithm is vector based equations, when the body temperature is calculated and body heat radiation of each penguin and then due to distance and attractiveness each penguin performs the spiral-like movement.

EPC starts with a set of penguins representing the population size. These penguins are distributed in nature with calculated position and cost, penguins are continually moving in the direction of low objective value penguins, these penguins with high intensity. The objective function value is calculated using heat intensity and the distance. Attraction is done, a new solution is evaluated and the heat intensity is updated. All solutions are sorted and the best is selected. Damping ratio for heat radiation, movement, and heat absorption is applied. Figure 2 describes pseudo code of the EPC algorithm.

```
Generate Initial Population Array of EPs (Colony Size);
Generate Position of Each EP;
Generate Cost of Each EP; Determine Initial Heat Absorption
Coefficient;
For It= 1to Maxiteration Do
Generate Repeat Copies of Population Array;
For I= 1 To N Population Do
 For J=1to N Population Do
   If Costj < Costi Then
    Calculate Heat Radiation (Eq. 4);
   Calculate Attractiveness (Eq. 5);
   Calculate Coordinated Spiral Movement;
   Determine New Position;
   Evaluate New Solutions;
  End
 End
End
 Sort and Find Best Solution;
 Update Heat Radiation (Decrease);
 Update Mutation Coefficient (Decrease);
Update Heat Absorption Coefficient (Increase);
End
```

Figure 2. Pseudo code of the EPC algorithm [30]

This algorithm is performed according to the following rules [30]:

1- All penguins in the initial population have heat radiation and attract to each other due to absorption coefficient.

2- The body surface area of all penguins is considered equal to each other.

3- Penguin absorbs the full heat radiation and the effect of the earth's surface and the atmosphere are not regarded.

4- The heat radiation of penguins is considered linear.

5- The attraction of penguin is done according to the amount of heat in the distance between two penguins.

6- The penguin spiral movement during the absorption process is not monotonous and has a deviation with uniform distribution.

This algorithm has several of advantages, namely, simplicity, ease of implementation, providing a solution to complex problems and not requiring large population size to start solving any optimization problem [31].

The heat radiation of each penguin is calculated using Eq. (4),

$$Q_{penguin} = A\varepsilon\,\sigma\,T_s^4 \quad (4)$$

where, Q_{penguin} is heat transfer per unit of time, A is total surface area of the penguin which equal to 0.56 m^2. ε is emissivity of bird plumage which is considered 0.98, σ is the Stefan–Boltzmann constant (5.6703×10−8 W/m2K4) and Ts is the absolute temperature in Kelvin (K) which is considered 35°C equals to 308.15 K [30].

The attractiveness Q is calculated using Eq. (5),

$$Q = A\varepsilon\,\sigma\,T_s^4\,e^{-\mu x} \quad (5)$$

where, μ is attenuation coefficient and x are the distance between two linear sources.

The calculation of the coordinated spiral movement and the new position is done using logarithmic spiral movement in the original EPC [30].

5.3 First Modified Emperor Penguins Colony (MEPC1)

Two modifications are performed to the original emperor penguin colony algorithm (EPC) introduced by Harifi et al. [30], the two modifications are called MEPC1 and MEPC2, MEPC1 is developed by Wahdan et al. [31]. MEPC1 involves changing the algorithms used to represent the spiral movement of the EPC. In the original EPC given in the ref. [30], the penguins colony move from a cold environment to a warmer environment using logarithmic like spiral movement. In MEPC1, the penguins colonies move from position i to position j using Archimedes spiral like movement.

As shown in Figure 3, Suppose there are two penguins i and j. Moving always is from the penguin that needs heat to the penguin that is warmer. Here the spiral movement is from i to j, because in this case the penguins j is warmer, penguin i starts movement form position i and is attracted to position j using spiral like movement. To reach position j, the penguins should reach new position(s) k.

MEPC1 algorithm has proven successful in solving continuous optimization problems compared to the original EPC. The algorithm is efficient for solving complex problems in terms of solution quality and convergence rate [31].

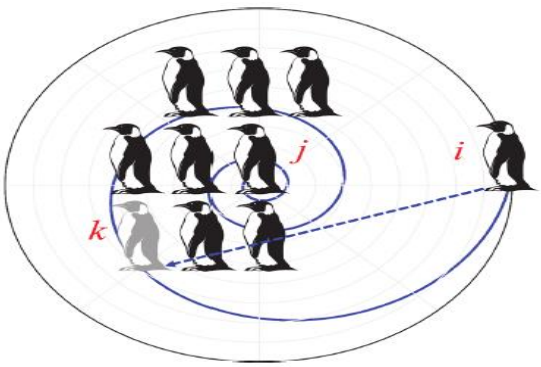


Figure 3. Spiral like movement of emperor penguins

5.4 Second Modified Emperor Penguins Colony (MPC2) Algorithm

The second modification performed to the original EPC is developed by Wahdan et al. [31]. The modification assumes hyperbolic spiral like movement instead of logarithmic like spiral movement.

When compared to other eight metaheuristics, MEPC2 achieved better results in most cases, in terms of objective function value [31].

5.5 Implementation

5.5.1 Solution representation

The four proposed algorithms (EPC, MEPC1, MEPC2 and CS) are designed mainly to solve continuous optimization problems. The problem under study is considered discrete optimization problem because a solution (nests or penguins) is represented by a vector of length that is equal to the number of elements in DSM. Each cell in the vector can assume values from 1 which represent the lower limit and to the size of the DSM which represent the upper limit.

As shown in Figure 4, a vector represents a solution to the problem, where the DSM size is 10 components with specific interaction between each other; these components want to be assigned in specific cluster, the vector in Figure 4 represents one of the solutions of the problem, and shows that we have three clusters. Cluster number one contains components 1, 2, 10, cluster two contains components 3, 5, 6, 7 and finally the third cluster contains components 4, 8, 9. These numbers in the vector are between 1 and 10, 1 represents the lower limit and 10 represent the upper limit. We start with number of cluster equal number of components in product, and optimization process performed to find the optimal number of clusters after deleting empty clusters.

1	1	2	3	2	2	2	3	3	1

Figure 4. Example of Solution representation

The problem presented in this work is discrete in nature, and hence, the proposed algorithm requires a process of discretization. Several methods are available in the literature to perform discretization. Among the known discretization methods is the random key technique, where continuous values are transformed into discrete integer values [32]. Another method is the smallest position value (SPV) [33]. A different technique available in the literature is the nearest integer (NI) method. In this method, a continuous valueis trasnformed to the nearest integer value by simply rounding, trancatingup, or trancatingdown [34].

SPV and random key methods do not permit the repetition of integer values in the solution. Solving the problem presented in this research necessarily requires repeating some integer values. Hence, SPV and random key methods cannot be used to solve the problem in hand. The nearest integer method, on the other hand, allows the repetition of integer values in a solution, and therefore, the nearest integer discretization technique is chosen to solve the problem in this research. This discretization technique is efficient in solving many problems before [4, 5, 23].

5.5.2 Solution evaluation

Clustering a DSM requires minimizing the total coordination cost which is based on *IntraClusterCost* and *ExtraClusterCost*. *IntraClusterCost* is calculated if interaction *DSMik* belongs to cluster j, otherwise, *ExtraClusterCost* is calculated. At the beginning of the solution procedure, feasible solutions are randomly generated, and the total coordination cost is calculated. Details of calculating the total coordination cost, IntraClusterCost, and ExtraClusterCost, are given in

Section 4. Evaluation of the solution(s) is performed, then the algorithm selects the best obtained solution and a new iteration begins.

In this research, four algorithms are used, hence, each algorithm moves to the next solution according to its specific procedure as follows: when using CS to cluster a DSM, the algorithm begins with generating a set of nests. A nest consists of a vector having a length that is equal to the number of elements of the DSM to be clustered. Entries of vectors are randomly generated uniformly between the upper limit and lower limit. Then, these entries are transformed into integer values using the nearest integer method. Vectors represent solutions that require evaluation. Therefore, each vector is sent to the evaluation function. The evaluation function calculates the corresponding total coordination cost, conducts comparisons and performs updates using Levy flight and the probability of discovery (pa) (4).

Regarding EPC, MEPC1 and MEPC2, the algorithms start with a set of penguins representing the population size. These penguins consist of a vector having a length that is equal to the number of elements of the DSM to be clustered. Entries of vectors are randomly generated uniformly between the upper limit and lower limit. Then, these entries are transformed into integer values using the nearest integer method. Vectors represent solutions that require evaluation. Therefore, each vector is sent to the evaluation function. The evaluation function calculates the corresponding total coordination cost, all solutions are sorted and the best is selected. Damping ratio for heat radiation, movement, and heat absorption is applied. Each of the four algorithms is moved from generation to the next till the stopping criteria are reached.

6. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, the four algorithms are examined on different test instances. Three instances are given in the literature and 80 instances with different dimensions and complexities are randomly generated. Several parameters exist for the 4 algorithms. Parameters' setting for these algorithms are as follows: population size is 25 penguins in case of EPC, MEPC1 and MEPC2 and 25 nests in case of CS. The maximum number of iterations equal 100 in all algorithms. For EPC, MEPC1 and MEPC2 the mutation factor ϕ is set at 0.05 as recommended by (30). For CS, (pa) is a parameter that represents exploration. In the experimental setting of this research, the value of (pa) is set at 0.25 as recommended in (5).

	1	2	3	4	5	6	7
1		1	0	0	1	1	0
2	0		0	1	0	0	1
3	0	1		1	0	0	1
4	0	1	1		1	0	1
5	0	0	0	1		1	0
6	1	0	0	0	1		0
7	0	1	1	1	0	0	

Figure 5. Original DSM

	1	5	6	2	3	4	7
1		1	1	1	0	0	0
5	0		1	0	0	1	0
6	1	1		0	0	0	0
2	0	0	0		0	1	1
3	0	0	0	1		1	1
4	0	1	0	1	1		1
7	0	0	0	1	1	1	

Figure 6. Clustered DSM

The first small size instance has a DSM that contains 7 elements as shown in Figure 5 [9], after applying EMP, MEPC1, MEPC2 and CS algorithms. Results obtained are given in Figure 6. Figure 6 shows that **two** clusters are formed, elements 1, 5, 6 are assigned to cluster 1, elements 2, 3, 4, 7 are assigned to cluster 2 and objective function value is 47.29. The four algorithms provide the same results.

Another example with 9 elements is used to test the efficiency of the algorithms [26]. The original DSM is given in Figure 7, and results obtained are given in Figure 8. Figure 8 shows that **four** clusters are formed, elements A, E, G are assigned to cluster 1, elements B, C, F, H are assigned to cluster 2, element D is assigned to cluster 3 and element I is assigned to cluster 4. The four algorithms provide the same results with Objective function value of 41.8.

	A	B	C	D	E	F	G	H	I
A					1		1		
B			1			1		1	
C		1				1		1	
D									
E	1						1		
F		1	1					1	
G	1				1				
H		1	1			1			
I									

Figure 7. Original DSM

	A	E	G	B	C	F	H	D	I
A		1	1						
E	1		1						
G	1	1							
B					1	1	1		
C				1		1	1		
F				1	1		1		
H				1	1	1			
D									
I									

Figure 8. Clustered DSM

A large size instance available in the ref. [25] is also used to examine the proposed algorithms. The instance represents an elevator example, and the objective function value obtained in (24) equals 4433. After solving the elevator example using the proposed algorithms, the following results are obtained: CS obtained an objective function value of 4133.25, EPC obtained an objective function value of 4108.714, MEPC1 obtained an objective function value of 4090.51, and MEPC2 obtained an objective function value of 4079.58. Hence, MEPC2 yields the best solution for the elevator example.

Due to the limited benchmark problems available in the literature for the problem in hand, 80 DSMs are randomly generated. These matrices contain 1's and 0's. The number of entries equals to 1 represents the existence of interaction, and they represent the problem's complexity. These matrices range from size 10 (number of elements in DSM) up to 100, and from complexity 0.2 up to 0.9. Complexity is defined as the ratio between the numbers of actual interactions to the total number of possible interactions in any given DSM. For the sake of comparison in various experiments, each of the four proposed algorithms set for 30 runs, 100 iterations, and population size of 25.

Results obtained are given in the Tables 1-10. Table 1 shows the results obtained in case of DSM of size 10 elements, MEPC2 provide the best solution in 7 out of 8 problems. The best minimum value is written in bold.

Table 2 gives the results obtained in case of DSM with size 20 elements, MEPC2 provides the best solution in 6 out of 8 problems, and EPC provides the best solution in the remaining 2 problems.

Table 3 shows the results obtained in case of DSM with size 30 elements, EPC yields the best solution in 6 out of 8 problems, and MEPC2 provides the best solution in the remaining 2 problems.

Table 4 provides the results obtained in case of DSM with size 40 elements, EPC provide the best solution in all test instances.

Table 1. Results obtained in case of 10 elements

Problem NO.	Complexity	EPC	MEPC1	MEPC2	CS
1	0.20	59.26	59.29	**59.11**	59.64
2	0.30	90.92	91.55	**90.86**	92.74
3	0.40	124.17	124.73	**124.17**	126.14
4	0.50	141.62	143.12	**141.35**	143.85
5	0.60	182.58	181.76	**181.78**	185.71
6	0.70	222.79	222.08	**221.88**	226.07
7	0.80	262.30	**260.72**	260.95	266.94
8	0.90	301.50	298.83	**298.74**	305.37

Table 2. Results obtained in case of 20 elements

Problem NO.	Complexity	EPC	MEPC1	MEPC2	CS
9	0.20	410.74	410.12	**409.84**	448.57
10	0.30	633.49	635.20	**633.12**	695.21
11	0.40	845.67	848.92	**845.38**	930.58
12	0.50	1025.98	1031.18	**1025.55**	1131.10
13	0.60	1268.62	1274.23	**1267.94**	1394.53
14	0.70	1501.73	1514.06	**1500.38**	1653.73
15	0.80	**1742.36**	1751.44	1756.81	1912.14
16	0.90	**1981.62**	1995.36	1992.33	2170.11

Table 3. Results obtained in case of 30 elements

Problem NO.	Complexity	EPC	MEPC1	MEPC2	CS
17	0.20	1243.77	1244.20	**1240.92**	1416.35
18	0.30	1892.66	1888.89	**1887.01**	2156.63
19	0.40	**2452.75**	2460.71	2458.43	2800.52
20	0.50	**3137.64**	3150.20	3144.74	3574.28
21	0.60	**3823.16**	3840.35	3840.50	4356.61
22	0.70	**4510.58**	4539.36	4537.98	5136.24
23	0.80	**5201.03**	5237.51	5242.36	5916.99
24	0.90	**5892.03**	5942.50	5945.68	6692.67

Table 4. Results obtained in case of 40 elements

Problem NO.	Complexity	EPC	MEPC1	MEPC2	CS
25	0.20	2659.34	2688.08	2688.04	3110.47
26	0.30	4108.99	4145.67	4153.25	4793.71
27	0.40	5374.10	5430.81	5436.64	6274.72
28	0.50	6833.19	6901.61	6924.44	7960.46
29	0.60	8286.32	8381.90	8389.70	9654.20
30	0.70	9754.97	9867.63	9893.70	11349.13
31	0.80	12695.63	12854.96	12863.97	14737.86
32	0.90	12695.56	12854.79	12841.51	14716.40

Table 5 provides the results obtained in case of DSM with size 50 elements, MEPC2 provides the best solution in all test instances.

Table 6 shows the results obtained in case of DSM with size 60 elements, MEPC2 provide the best solution in all test instances.

Table 7 provides the results obtained in case of DSM with size 70 elements, EPC provides the best solution in all test instances.

Table 5. Results obtained in case of 50 elements

Problem NO.	Complexity	EPC	MEPC1	MEPC2	CS
33	0.20	4872.99	4876.53	**4864.79**	5696.09
34	0.30	7554.15	7555.88	**7549.51**	8813.82
35	0.40	10162.66	10150.55	**10136.22**	11835.57
36	0.50	12644.24	12633.26	**12594.36**	14694.53
37	0.60	15289.76	15304.14	**15277.54**	17788.42
38	0.70	17993.41	17961.41	**17936.48**	20855.22
39	0.80	20669.70	20642.17	**20633.62**	23953.84
40	0.90	23357.20	23331.84	**23317.28**	27031.44

Table 6. Results obtained in case of 60 elements

Problem NO.	Complexity	EPC	MEPC1	MEPC2	CS
41	0.20	8241.80	8124.36	**8120.49**	9563.00
42	0.30	12502.10	12339.54	**12331.14**	14516.15
43	0.40	16467.73	16317.96	**16308.94**	19150.40
44	0.50	20845.65	20624.13	**20628.78**	24177.89
45	0.60	25181.64	24938.64	**24912.80**	29206.28
46	0.70	29511.19	29265.34	**29228.18**	34227.98
47	0.80	33886.56	33588.34	**33565.04**	39268.96
48	0.90	38153.21	37935.76	**37926.04**	44266.25

Table 7. Results obtained in case of 70 elements

Problem NO.	Complexity	EPC	MEPC1	MEPC2	CS
49	0.20	**12278.74**	12388.62	12521.29	14662.28
50	0.30	**18047.36**	18241.68	18415.55	21553.46
51	0.40	**24429.92**	24699.62	24877.97	29149.01
52	0.50	**30813.92**	31164.59	31339.48	36730.60
53	0.60	**37200.45**	37629.73	37839.29	44338.85
54	0.70	**43605.67**	44144.83	44415.53	51914.27
55	0.80	**50024.42**	50648.82	50870.36	59507.85
56	0.90	**56462.69**	57172.20	57357.32	67122.44

Table 8. Results obtained in case of 80 elements

Problem NO.	Complexity	EPC	MEPC1	MEPC2	CS
57	0.20	**17525.86**	17749.66	17898.10	21063.75
58	0.30	**26259.91**	26574.60	26728.03	31517.85
59	0.40	**34918.14**	35395.07	35505.06	41883.51
60	0.50	**44020.18**	44620.30	44742.08	52743.66
61	0.60	**53128.16**	53814.11	53995.54	63606.21
62	0.70	**62230.02**	63063.00	63233.70	74434.30
63	0.80	**71355.15**	72318.92	72423.36	85317.54
64	0.90	**80514.41**	81606.40	81709.90	96163.83

Table 9. Results obtained in case of 90 elements

Problem NO.	Complexity	EPC	MEPC1	MEPC2	CS
65	0.20	**24075.11**	24283.99	24383.92	28879.44
66	0.30	**36262.51**	36570.27	36728.63	43434.52
67	0.40	**48085.50**	48547.90	48629.22	57602.77
68	0.50	**60507.28**	61114.71	61271.23	72469.39
69	0.60	**72976.99**	73738.22	73855.36	87363.44
70	0.70	**85388.53**	86332.14	86468.65	102199.42
71	0.80	**97831.44**	98974.95	99099.94	117038.63
72	0.90	**110266.79**	111648.85	111760.94	131906.45

Table 10. Results obtained in case of 100 elements

Problem NO.	Complexity	EPC	MEPC1	MEPC2	CS
73	0.20	32358.58	**32231.73**	32253.43	38386.95
74	0.30	47918.90	**47820.50**	47828.16	56879.90
75	0.40	64576.49	**64427.24**	64460.62	76554.42
76	0.50	81195.57	**81094.70**	81116.32	96239.47
77	0.60	97831.45	**97738.73**	97727.19	115896.80
78	0.70	114485.53	**114454.57**	114445.83	135621.28
79	0.80	131159.43	**131109.21**	131117.61	155255.84
80	0.90	147849.93	147819.83	**147780.66**	174927.68

Table 8 provides the results obtained in case of DSM with size 80 elements, EPC provide the best solution in all test instances.

Table 9 provides the results obtained in case of DSM with size 90 elements, EPC provide the best solution in all test instances.

Table 10 provides the results obtained in case of DSM with size 100 elements, MEPC1 provides the best solution 7 out of

8 problems.

Results given in Tables 1 to 10 shows that, EPC outperforms the other three proposed algorithms in 50% of the tested problems, MEPC1 outperforms the other 3 proposed algorithms in 10% of the tested problems and finally MEPC2 outperforms the other 3 proposed algorithms in 40% of the tested problems.

Sensitivity analysis is performed to examine the effect of changing parameters on the quality of solutions. Regarding CS, there are two parameters that need setting, namely, the number of nests and the probability of discovery (Pa). The number of nests (n) represents the population size. Sensitivity analysis is performed by solving the same test instances using CS with different values of (n) to be 25, 50, 75 and 100. Sensitivity analysis shows that changing the number of nests does not have a significant impact on the objective function value for all tested cases. The second parameter (pa) represents the probability of discovery. The sensitivity analysis is performed by solving the test instances using CS at values of (pa) in the range [0.1, 0.9], with an increment of 0.1. Results show that high values of (pa) tend to get rid of solutions without trying to improve them locally. Therefore, the value of (pa) in the range [0.1, 0.5] for these test instances, achieves the required balance between exploration and exploitation.

Regarding EPC, MEPC1 and MEPC2, sensitivity analysis is performed for two parameters, namely, the number of penguins (colony size) and the mutation factor ϕ. The sensitivity analysis involved changing the number of penguins from 20 to 100 and the value of the mutation factor ϕ from 0.01 to 0.09. Several runs are performed on the same test instances mentioned above with different values of number of penguins and ϕ. These runs conclude that, changing the number of penguins or changing the value of the mutation factor does not have a significant effect on the results obtained.

The performance of the four algorithms is ranked using the Friedman test [35]. The Friedman test is a non-parametric method for identifying treatment discrepancies through several attempts. Table 11 shows the ranking of algorithms based on the results obtained from Table 1 to 10 (80 problems) using the Friedman test. Table 11 shows that EPC algorithm is first in the ranking; MEPC2 comes next, followed by MEPC1 and, finally CS.

Table 11. Ranking of algorithms based on performance using Friedman's test

Algorithms	EPC	MEPC1	MEPC2	CS
Ranking	1.83125	2.1375	2.03125	4

To find significant differences between the results obtained by the algorithms in solving the 80 test problems, statistical analysis is used. To detect significant differences in the results, Friedman test is employed. When applying Friedman test using the online Friedman calculator, the result is significance at $p < 0.05$. This means that the results are important.

Given The null hypothesis for the Friedman test is that there is no a significant difference between the results, Table 12 shows the results of the Friedman test The Chi-Square value is 146.32, with 3 degrees of freedom, and also there is asymptotic significance of the test (p-value) with very close to zero value. Given the close to zero value of the asymptotic significance, the hypothesis is rejected. Therefore, it can be concluded that there is a significant difference in the performance of algorithms.

Table 12. Results of Friedman's tests based on performance

Test method	Chi-Square	Degrees of freedom (DF)	P-value	Hypothesis
Friedman	146.32	3	*0.00001*	Rejected

7. CONCLUSION AND FUTURE WORK

This work provided a comparison between four population-based optimization algorithms. This is the first research that used EPC, MEPC1 and MEPC2 in solving discrete optimization problem. The research aimed to find the optimal number of clusters (modules) in DSM and the best assignment of each element to a specific cluster. A product is represented using DSM, DSM is a product representation tool; it provides graphical representation to the system elements. The objective function was to minimize the total coordination cost. The utilized algorithms were tested and compared on three problems found in literature and eighty test instances randomly generated. Results showed that the EPC algorithm was the first in the Friedman ranking test, MEPC2 comes next, followed by MEPC1, then CS. Future work may include multi-objective EPC algorithm to find the optimal assignment of each element in the cluster and the optimal number of clusters which minimize total coordination cost of the product and maximize product sustainability and provide hybridization between CS and EPC to solve discrete optimization algorithms.

REFERENCES

[1] Chang, T.R., Wang, C.S., Wang, C.C. (2013). A systematic approach for green design in modular product development. The International Journal of Advanced Manufacturing Technology, 68(9): 2729-2741. https://doi.org/10.1007/s00170-013-4865-5

[2] Shaik, A.M., Rao, V.K., Rao, C.S. (2014). Development of modular manufacturing systems—a review. The International Journal of Advanced Manufacturing Technology, 74: 1-4. http://doi.org/10.1007/s00170-014-6289-2

[3] Abdelsalam, H., Rasmy, M., Mohamed, H.G. (2014). A simulation-based time reduction approach for resource constrained design structure matrix. International Journal of Modeling and Optimization, 4(1): 51-55. http://doi.org/10.7763/IJMO.2014.V4.346

[4] Wahdan, H.G., Abdelsalam, H.M., Kassem, S.S. (2017). Product Modularization using cuckoo search. In Operations Research and Enterprise Systems. vol 695. Springer, Cham. http//doi.org/10.1007/978-3-319-53982-9_2

[5] Wahdan, H., Kassem, S.S., Abdelsalam, H. (2016). A Cuckoo search clustering algorithm for design structure matrix. 5th the International Conference on Operations Research and Enterprise Systems (ICORES 2016), Italy, Rome, pp. 36-43. http//doi.org/10.5220/0005693000360043

[6] Eppinger, S., Whitney, D., Smith, R., Gebala, D.A. (1994). A model based method for organizing tasks in product development. Research in Engineering Design, 6(1): 1-3. https://doi.org/10.1007/BF01588087

[7] Idicula, J. (1995). Planning for concurrent engineering.

Singapore: Gintic Institute Research report.
[8] Gutierrez, C.I. (1998). Integration analysis of product architecture to support effective team co-location. Cambridge: Masters thesis, Massachusetts Institute of Technology.
[9] Yassine, A.A., Yu, T.L., Goldberg, D.E. (2007). An information theoretic method for developing modular architectures using genetic algorithms. Research in Product Design, 18(2): 91-109. http//doi.org/10.1007/s00163-007-0030-1
[10] Sosa, M.E., Rowles, C.M. (2003). Identifying modular and integrative systems and their impact on design team interactions. ASME J Mech Des, 125(2): 240-252. https://doi.org/10.1115/1.1564074
[11] Fabrice, A., Steven, B.S., Henri, J.T. (2006). Design structure matrix flow for improving identifiaction and specification of modules. International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, pp. 1-13. Philadelphia, Pennsylvania, USA: ASME 2006.
[12] Borjesson, F. (2009). Improved output in modular function deployment using heuristics. International Conferance on Engineering Design, Stanford, USA, pp. 24-27.
[13] Liang, L.Y. (2009). Grouping decomposition under constraints for design/build life cycle in project delivery system. International Journal of Technology Management, 48(2): 168-187. https://doi.org/10.1504/IJTM.2009.024914
[14] van Beek, T.J., Erden, M.S., Tomiyama, T. (2010). Modular design of mechatronic systems with function modeling. Mechatronics, 20(8): 850-863. https://doi.org/10.1016/j.mechatronics.2010.02.002
[15] Pandremenos, J., Chryssolouris, G. (2012). A neural network approach for the development of modular product architectures. International Journal of Computer Integrated Manufacturing, 14(3): 1-8. https://doi.org/10.1080/0951192X.2011.602361
[16] Borjesson, F., Hölttä-Otto, K. (2012). Improved clustering algorithm for design structure matrix. ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, Chicago, IL, USA, pp. 1-10. https://doi.org/10.1115/DETC2012-70076
[17] Borjesson, F., Sellgren, U. (2013). Fast hybrid genetic clustering algorithm for design structure matrix. 25th International Conference on Design Theory and Methodology. Portland, Oregon, USA. https://doi.org/10.1115/DETC2013-12041
[18] Yang, Q., Yao, T., Lu, T., Zhang, B. (2014). An overlapping-based design structure matrix for measuring interaction strength and clustering analysis in product development project. IEEE Transactions on Engineering Management, 61(1): 159-170. http//doi.org/10.1109/TEM.2013.2267779
[19] Kim, S., Baek, J.W., Moon, S.K., Jeon, S. (2015). A new approach for product design by integrating assembly and disassembly sequence structure planning. Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, pp. 247-257. https://doi.org/10.1007/978-3-319-13359-1_20
[20] Qiao, L., Efatmaneshnik, M., Ryan, M., Shoval, S. (2017). Product modular analysis with design structure matrix using a hybrid approach based on MDS and clustering. Journal of Engineering Design, 28(6): 433-456. http://dx.doi.org/10.1080/09544828.2017.132585
[21] Sakao, T., Song, W., Matschewsky, J. (2017). Creating service modules for customising product/service systems by extending DSM. CIRP Annals, 66(1): 21-24. https://doi.org/10.1016/j.cirp.2017.04.107
[22] Ezzat, O., Medini, K., Boucher, X., Delorme, X. (2019). Product and service modularization for variety management. Procedia Manufactoring, 28: 148-153. https://doi.org/10.1016/j.promfg.2018.12.024
[23] Wahdan, H., Abdelslam, H., Abou-El-Enien, T., Kassem, S. (2019). Sustainable product design through non-dominated sorting cuckoo search. Journal Européen des Systèmes Automatisés, 52(5): 439-448. https://doi.org/10.18280/jesa.520502
[24] Kongsin, T., Klongboonjit, S. (2020). Machine Component Clustering with Mixing Technique of DSM, Jaccard Distance Coefficient and k-Means Algorithm. EasyChair Preprint no. 2296.
[25] Thebeau, R. (2001). Knowledge management of system interfaces and interactions for product development process. Massachusetts Institute of Technology.
[26] Borjesson, F., ltta-Otto, K.H. (2014). A module generation algorithm for product architecture based on component interactions and strategic drivers. Research in Engineering Design, 25(1): 31-51. https://doi.org/10.1007/s00163-013-0164-2
[27] Elbeltagia, E., Hegazyb, T., Grierso, D. (2005). Comparison among five evolutionary-based optimization algorithms. Advanced Engineering Informatics, 19: 43-53. https://doi.org/10.1016/j.aei.2005.01.004
[28] Yang, X., Deb, S. (2009). Cuckoo search via levy flights. The World Congress on Nature and Biologically Inspired Computing (NABIC '09). Coimbatore, India: IEEE, pp. 210-214. https://doi.org/10.1109/NABIC.2009.5393690
[29] Yang, X.S., Deb, S. (2010). Engineering optimisation by cuckoo search. Int J Math Model Numer Optim, 1(4): 330-343.
[30] Harifi, S., Khalilian, M., Mohammadzadeh, J., Ebrahimnejad, S. (2019). Emperor penguins colony: A new metaheuristic algorithm. Evolutionary Intelligence, 12(2): 211-226. https://doi.org/10.1007/s12065-019-00212-x
[31] Wahdan, H.G., Abdelslam, H.E., Abou-El-Enien, T.H., Kassem, S.S. (2020). Two-modified emperor penguins colony optimization algorithms. Revue d'Intelligence Artificielle, 34(2): 151-160. https://doi.org/10.18280/ria.340205
[32] Chen, H., Li, S., Tang, Z. (2011). Hybrid gravitational search algorithm with random-key encoding scheme combined with simulated annealing. International Journal of Computer Science and Network Security, 11(6): 208-217.
[33] Beckmann, D., Dagen, M., Ortmaier, T. (2018). A comparison of discretization methods for parameter estimation of nonlinear mechanical systems using extended kalman filter: Symplectic versus classical approaches. In: Madani K., Peaucelle D., Gusikhin O. (eds) Informatics in Control, Automation and Robotics. Lecture Notes in Electrical Engineering, vol 430. Springer, Cham. https://doi.org/10.1007/978-3-319-55011-4_18
[34] Burnwal, S., Deb, S. (2012). Scheduling optimization of

flexible manufacturing system using cuckoo search-based approach. International Journal of Advanced Manufacturing Technology, 64: 1-9. https://doi.org/10.1007/s00170-012-4061-z

[35] Derrac, J., García, S., Molina, D., Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation, 1(1): 1-13. https://doi.org/10.1016/j.swevo.2011.02.002