Contents lists available at GrowingScience

Decision Science Letters

homepage: www.GrowingScience.com/dsl

A dynamic programming–enhanced simulated annealing algorithm for solving bi-objective cell formation problem with duplicate machines

Mohammad Mohammadi[*] **and Kamran Forghani**

Department of Industrial Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran

C H R O N I C L E

Article history:
Received June 10, 2014
Accepted October 22, 2014
Available online
October 22 2014

Keywords:
Cellular manufacturing
Hybrid simulated annealing
Dynamic programming
Machine duplication
Job shop
Flow shop

A B S T R A C T

Cell formation process is one of the first and the most important steps in designing cellular manufacturing systems. It consists of identifying part families according to the similarities in the design, shape, and presses of parts and dedicating machines to each part family based on the operations required by the parts. In this study, a hybrid method based on a combination of simulated annealing algorithm and dynamic programming was developed to solve a bi-objective cell formation problem with duplicate machines. In the proposed hybrid method, each solution was represented as a permutation of parts, which is created by simulated annealing algorithm, and dynamic programming was used to partition this permutation into part families and determine the number of machines in each cell such that the total dissimilarity between the parts and the total machine investment cost are minimized. The performance of the algorithm was evaluated by performing numerical experiments in different sizes. Our computational experiments indicated that the results were very encouraging in terms of computational time and solution quality.

1. Introduction

In today's competitive market, manufacturing industries must be able to produce products with low production cost and high quality to deliver the products to customers on time. In addition, they should be able to respond quickly to changes in product design and demand with low cost. Traditional manufacturing systems, such as job shops and flow shops cannot provide the efficiency and flexibility simultaneously to adapt to such requirements (Kioon et al., 2009). As a result, cellular manufacturing (CM), an application of group technology (GT), has emerged to satisfy such production requirements in manufacturing systems producing medium-volume/medium-variety products. GT is a manufacturing philosophy that identifies and explores the similarities of product design and manufacturing process to decompose a manufacturing system into several subsystems for facilitating the production floor control. In fact, CM is a hybrid manufacturing system that joints the advantages of flow shops and job shops with characteristics such as reduced cycle times compared to jobs shops

* Corresponding author. Tel. & Fax: +98 21 88830891
E-mail address: Mohammadi@khu.ac.ir (M. Mohammadi)

doi: 10.5267/j.dsl.2014.10.002

and increased flexibility and greater job satisfaction as compared with flow shops (Mungwattana, 2000). If a cellular manufacturing system (CMS) is designed properly, it leads to shorter cycle times, reduced material handling cost, reduced work-in-process inventories, reduced setup times, reduced tool requirements, better control over manufacturing processes and improved quality of products (Wemmerlöv & Hyer, 1989; Jeon & Leep, 2006). To implement an efficient CMS, several issues such as cell formation (CF), facility layout, production planning, scheduling, etc. should be considered in the design process. Among these issues, CF process is one of the first and most important steps in designing CMSs. It consists of identifying part families according to the similarities in the design, shape, and presses of parts and dedicating machines to each part family based on the operations required by the parts. This process is also addressed as the machine-part grouping problem (MPGP) in the literature. In the context of the CF problem, numerous approaches have been developed in the literature. Among those approaches, classification and coding systems, similarity coefficient-based methods, array-based clustering methods, mathematical programming, graph partitioning, artificial intelligence (AI) based methods and heuristic and meth-heuristic based algorithms are the frequently addressed approaches (Singh, 1993; Mungwattana, 2000).

As CF problems belong to the class of NP-hard combinational problems, most researchers have focused on implementation of meta-heuristic algorithms such as genetic algorithm (GA), simulated Annealing (SA), tabu search (TS), ant colony optimization (ACO), particle swarm optimization (PSO), etc. For instance, Boctor (1991) presented a 0-1 linear mathematical model for MPGP and aimed to minimize the number of exceptional elements (EEs). A SA algorithm was developed to solve the model. The proposed SA seems to able to find the optimal solution for 58 (64.4%) of the 90 solved problems. Chen and Srivastava (1994) formulated the CF problem as a quadratic programming model to maximize the total similarity between machines, subject to cell size limitation. They employed SA to solve the problem and concluded that the performance of the SA is better than the graph-partitioning heuristic. Balakrishnan and Jog (1995) proposed a parallel genetic TSP algorithm to minimize the number of EEs in the MPGP. In this study, the MPGP was represented as two TSPs by converting the similarity coefficients between the parts and between the machines into distances. Caux et al. (2000) addressed the problem of manufacturing cell formation with alternative processing routings, machine capacities, operation times and part demands. A hybrid approach combining the SA algorithm for the CF and a branch-and-bound (B&B) method for the routing selection was presented to minimize the number of intercellular movements. As the proposed algorithm uses the B&B method, it may not be efficient in solving large-sized problems or unconstrained problems both in terms of accuracy and computational time. Adil and Rajamani (2000) presented anon-linear mathematical model to investigate the trade-off between cell compactness and cell independence in terms of inter-cell and intra-cell move costs. A SA algorithm was used to solve the problem. Mungwattana (2000) studied the CF problem with alternative processing routings in dynamic and stochastic production requirements and utilized SA algorithm to solve the problem. Onwubolu and Mutingi (2001) developed a GA to solve the MPGP with two objectives, minimizing the total number of EEs and minimizing the total cell load-variation. The proposed GA appears to perform better than TSP-based heuristics. Solimanpur et al. (2004) presented a mathematical model for designing independent manufacturing cells with multiple objectives of minimizing the total dissimilarity, total processing cost, total processing time and total investment in the acquisition of machines. A GA with multiple fitness functions was used to find multiple solutions along the Pareto optimal frontier. Chiang and Lee (2004) developed a SA algorithm augmented with dynamic programming to solve the CF problem with an objective function of minimizing the inter-cell handling cost. Tavakkoli-Moghaddam et al. (2005) employed GA, SA and TS to solve a modified version of proposed problem by Mungwattana (2000). Their computational experiments indicated the superiority of SA over GA and TS. In addition, Tavakkoli-Moghaddam et al. (2008) modified the same work by considering reconfiguration and employed the SA algorithm to simultaneously minimize the inter-cell movements and machine costs. Their computational experiments showed that the gap between optimal and SA solutions is less than 4%. Arkat et al. (2007) developed a sequential CF model and solved it by SA and GA. They reported similar results for both methods. However, SA needed less computational time. (Saghafian & Akbari Jokar, 2009) extended the work of Chiang and Lee (2004) to intra-cell layout and developed a hybrid method based on SA, ACO and dynamic programming to minimize the total inter-cell and intra-cell handling cost. Banerjee and Das (2012) defined a new grouping efficacy in terms of a linear combination between the operation densities within the cells and the number of inter-cellular movements. A predator-prey GA was employed to solve the MPGP by considering this measure. Ghezavati and Saidi-Mehrabad (2011) applied queuing theory to formulate the CF problem with stochastic parameters. They assumed that each machine works as a server and each part is a customer and aimed to maximize the average probability that machines are busy in cells. A hybrid method based on combination of SA and GA was proposed to solve the problem efficiently. This hybrid method was compared against global solution obtained from B&B algorithm and a benchmark heuristic algorithm. They reported successful performance in any size of problems. Kao and Lin (2012) proposed a discrete PSO algorithm to minimize the number of EEs in the CF problem. They concluded that the PSO algorithm results in better solutions in comparison with the SA and TS-based algorithms.

In spite of many researches on the CF problem, most of the existing methods or algorithms in this area are not general enough to determine the number of machines in the cell design process. These approaches usually start with an initial machine-part matrix and then rearrange the columns and rows of this matrix to form part families and machine cells. In other words, those methods do not consider the capacity of machines and the processing information of parts such as demand and processing times in the cell design process. In this study, a bi-objective mathematical model in CMSs is presented to design independent manufacturing cells considering duplicate machines. The proposed model is to determine the part families and the number of machines in each cell such that the total dissimilarity between the parts and the total machine investment cost are minimized. Due to the complexity and NP-hard nature of CF problems, a hybrid solution approach based on combination of SA algorithm and dynamic programming is developed to solve the problem efficiently. After setting the parameters of the proposed algorithm, its performance is evaluated by performing numerical experiments in different sizes.

2. Model description

In our problem, parts are grouped into the part families based on the dissimilarity measure defined between them and considering the maximum number of parts allowed in a cell. In addition, machines are allocated to each part family such that the demand of all parts are satisfied and all the operations required by a part family are processed within a cell (i.e., the EEs are eliminated by duplicating machines). Two conflicting objective functions are considered in the problem. The first objective function, which minimizes the total dissimilarity between the parts are used to control the flow line-ness of the system. From the other side, the second objective function is used to control the job shop-ness of the system by minimizing the total machine investment cost. In other words, the designer will be able to adjust the job shop-ness or flow shop-ness of the manufacturing system in terms of the weight of each objective in the problem.

2.1. Notations

The following notations are used in the formulation of the problem.

Sets:

i, j	parts index $i, j = 1, \ldots, P$ (P is the number of parts)
k	machines index $k = 1, \ldots, M$ (M is the number of machine types)
l	cells index $l = 1, \ldots, C^{max}$ (C^{max} is the maximum number of cells allowed)

Parameters:

d_i	demand of part i
A_k	available time of machine type k
c_k	purchase price of machine type k
a_{ik}	=1 if part i visits machine k; 0 otherwise
t_{ik}	processing time of part i on machine type k
s_{ij}	dissimilarity coefficient between parts i and j
NP	maximum number of parts permissible in a cell (cell size limit)
TD	total dissimilarity between parts
TI	total machine investment cost
(w_1, w_2)	weights of TD and TI, respectively

Decision variables:

z_{il}	=1 if part i is assigned to cell l; 0 otherwise
y_{kl}	number of machines of type k assigned to cell l

2.2. Problem formulation

In context of similarity and dissimilarity coefficients, extensive reviews can be found in (Yin & Yasuda, 2005; Yin & Yasuda, 2006; Garbie et al., 2008). Among various similarity coefficients, Jaccard similarity coefficient proposed by McAuley (1972) is the most stable and most often used similarity coefficient in the literature Yin and Yasuda (2005). It is also a very simple and effective measure in designing CMSs. In this study, the dissimilarity coefficient between two parts i and j is defined by subtracting Jaccard similarity coefficient from its upper bound of 1, and it is defined as follows:

$$s_{ij} = 1 - \frac{\sum_k a_{ik} a_{jk}}{\sum_k a_{ik} + a_{jk} - a_{ik} a_{jk}}, \qquad \forall\, i, j. \tag{1}$$

Now, the proposed bi-objective problem can be formulated as the following non-linear integer model:

$$\min TD = \sum_{\substack{j>i \\ l}} s_{ij} z_{il} z_{jl}, \tag{2.1}$$

$$\min TI = \sum_{k,l} c_k y_{kl}. \tag{2.2}$$

Subject to:

$$\sum_l z_{il} = 1, \qquad \forall\, i, \tag{3}$$

$$\sum_i z_{il} \le NP, \qquad \forall\, l, \tag{4}$$

$$y_{kl} \ge \frac{\sum_i d_i t_{ik} z_{il}}{A_k}, \qquad \forall\, k, l, \tag{5}$$

$$z_{il} \in \{0,1\}, \qquad \forall\, i, l, \tag{6}$$

$$y_{kl} \text{ is integer}, \qquad \forall\, k, l. \tag{7}$$

Objective function (1.1) minimizes the total dissimilarity and objective function (2.2) minimizes the total machine investment cost. Constraint (3) ensures that each part is assigned to one cell. Constraint

(4) ensures that the cell size limit is not violated. Constraint (5) determines the number of each machine type in each cell. Lastly, constraints (6) and (7) indicate the type of decision variables.

The mathematical model presented above is non-linear, due to the presence of the product term $z_{il}z_{jl}$ in objective function (2.1). Most non-linear problems are usually much harder to solve optimally than linear problems. Here a method from Kaufmann and Broeckx (1978) which has the smallest number of variables and constraints is applied to linearize the model. To do this, objective function (2.1) is rearranged as follows: $\sum_{i,l} z_{il}\left(\sum_{j>i} s_{ij}z_{jl}\right)$. Now, by introducing a new set of variables γ_{il} to replace the $\sum_{j>i} s_{ij}z_{jl}$ product terms, and also defining constraints (9) and (10), objective function (2.1) is linearized as follows:

$$\min TD = \sum_{i,l} \gamma_{il}. \tag{8}$$

Subject to:

$$\gamma_{il} \geq \sum_{j>i} s_{ij}z_{jl} + (z_{il} - 1)\sum_{j>i} s_{ij}\,, \qquad \forall\, i, l, \tag{9}$$

$$\gamma_{il} \geq 0, \qquad \forall\, i, l. \tag{10}$$

To convert the proposed bi-objective model into a single objective one, a weighted sum of objective is used as follows:

$$\min w'_1\left(\sum_{i,l} \gamma_{il}\right) + w'_2\left(\sum_{k,l} c_k y_{kl}\right). \tag{11}$$

Subject to*:* (3)–(7), (9) and (10).

In objective function (11), parameters w'_1 and w'_2 are defined as follows:

$$w'_1 = \frac{w_1}{TD_{\mathrm{U}} - TD_{\mathrm{L}}}, \qquad w'_2 = \frac{w_2}{TI_{\mathrm{U}} - TI_{\mathrm{L}}}. \tag{12}$$

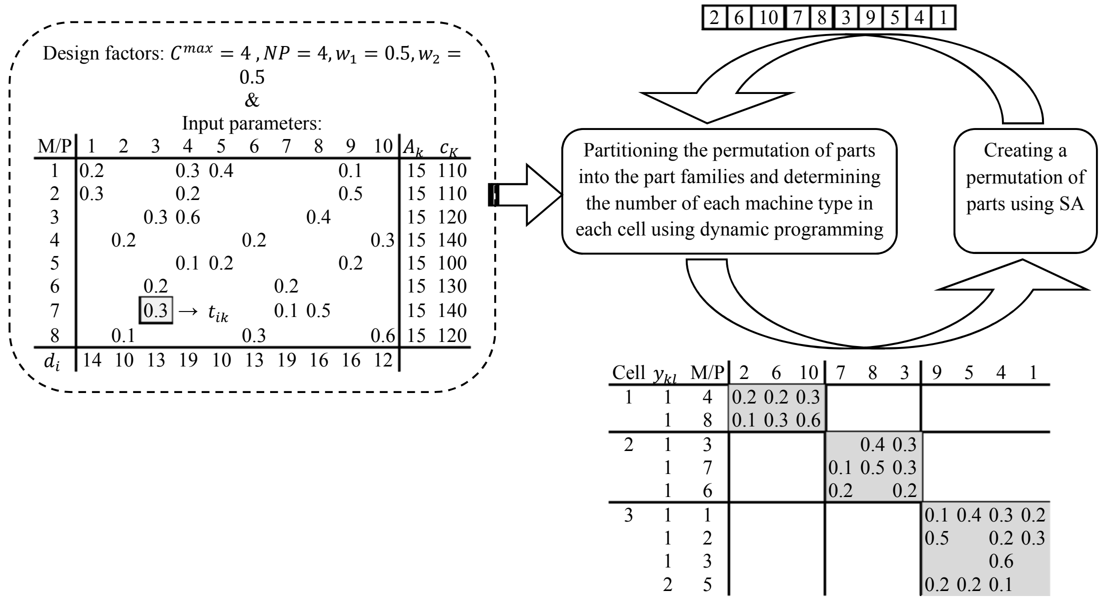
where TD_{L} and TD_{U} respectively denote the lower and upper bounds of TD, and TI_{L} and TI_{U} respectively shows the lower and upper bounds of TI. To determine TD_{L} and TI_{U} we can solve the problem with the following objective function: $\min \varepsilon \times TD + TI$ (Where ε is a small enough number). Also, TD_{U} and TI_{I} can be obtained by solving the problem with the following objective function: $\min TD + \varepsilon \times TI$.

3. Proposed enhanced SA algorithm

SA is a stochastic search method, which imitates the physical annealing of solid for solving combinatorial optimization problems. It has been known that the CF problem is one of the NP-hard combinational problems (Kazerooni et al., 1997). It means that obtaining optimal solution in a reasonable computational time is difficult, especially for large-scale problems. In recent years, SA algorithms have been successfully applied by researchers for solving the CF problems (Chiang & Lee, 2004; Tavakkoli-Moghaddam et al., 2005; Arkat et al., 2007; Ghosh et al., 2011). These have motivated us to employ the SA algorithm for solving the proposed problem. In addition, a dynamic programming algorithm is developed to improve the performance of the SA. The main elements of the proposed solution approach are explained below.

3.1. Generating initial solution

In the SA implementation, each solution must be represented by a coding scheme. In this work, a string of randomly generated integer values is used to represent a solution. If a problem involves P parts, a permutation of integer values with the length of P bits is needed to encode the solution. This string is associated with the sequence of parts at the machine-part matrix, which must be partitioned into part families and machine cells. In this way, a dynamic programming algorithm is developed to solve the partitioning problem. It should be noted that applying this coding scheme yields only feasible solutions and also leads to the reduction of the string length needed to represent a solution Chiang and Lee (2004). The proposed solution approach has been illustrated in Fig. 1.


2 6 10 7 8 3 9 5 4 1
Design factors: $C^{max} = 4$, $NP = 4, w_1 = 0.5, w_2 = 0.5$
&
Input parameters:
$\boxed{0.3} \rightarrow t_{ik}$
Partitioning the permutation of parts into the part families and determining the number of each machine type in each cell using dynamic programming
Creating a permutation of parts using SA

M/P	1	2	3	4	5	6	7	8	9	10	A_k	c_K
1	0.2			0.3	0.4				0.1		15	110
2	0.3			0.2					0.5		15	110
3			0.3	0.6				0.4			15	120
4		0.2				0.2				0.3	15	140
5				0.1	0.2				0.2		15	100
6			0.2				0.2				15	130
7			0.3				0.1	0.5			15	140
8		0.1				0.3				0.6	15	120
d_i	14	10	13	19	10	13	19	16	16	12		

Cell	y_{kl}	M/P	2	6	10	7	8	3	9	5	4	1
1	1	4	0.2	0.2	0.3							
	1	8	0.1	0.3	0.6							
2	1	3					0.4	0.3				
	1	7				0.1	0.5	0.3				
	1	6				0.2		0.2				
3	1	1							0.1	0.4	0.3	0.2
	1	2							0.5		0.2	0.3
	1	3									0.6	
	2	5							0.2	0.2	0.1	

Fig. 1. Proposed hybrid solution approach

3.2. Evaluating solutions by dynamic programming

The proposed scheme for encoding the solutions gives only the permutation of parts in the machine-part matrix and not the part families and the number of machines required in each cells. As it has been assumed that the machine cells must be formed independently (i.e., no material movement is allowed between the cells), the partition problem can be solved by dynamic programming algorithm. Here, a dynamic programming algorithm is presented to partition the permutation of parts into the part families and determine the number of machines in each cell under the cell size limit (NP) and the maximum number of cells (c^{max}) constraints.

3.2.1 Dynamic programming algorithm

In this section, we show how a dynamic programming approach can be used to solve the partition problem. Let $\overline{\pi}$ denotes a permutation of P parts. Now, the partition problem can be stated as follows. In a permutation of P parts, a breaking node is the part, which forms a part family (or one cell) and the node after the breaking node is the beginning for forming the next part family. Let b_l be the order index of node l in permutation $\overline{\pi}$, and $\pi(b_l)$ denotes the part index which has placed in order b_l. The partition problem is to find a set of L breaking nodes ($L \leq c^{max}$) which partition the permutation of parts into L part families in such a way that the weighted sum of the total dissimilarity and the total machine investment cost are minimized. Based on these definitions, the partition problem can be formulated as the following integer programming model:

$$\min TC(\overline{\pi}) = \sum_{l=1}^{L} w'_1 S_{b_{l-1},b_l} + w'_2 B_{b_{l-1},b_l}. \tag{13}$$

subject to:

$$1 \le b_1 < \cdots < b_L = P, \tag{14}$$

$$b_l - b_{l-1} \le NP, \qquad \forall\, l = 1, \dots, L, \tag{15}$$

$$L \le c^{max}, \tag{16}$$

$$b_0 = 0, \tag{17}$$

where $S_{b_l,b_{l-1}}$ calculates the increased dissimilarity from breaking node $b_{l-1}+1$ to breaking node b_l, and $B_{b_l,b_{l-1}}$ calculates the increased machine investment cost from breaking node $b_{l-1}+1$ to breaking node b_l. These parameters are calculated as follows:

$$S_{b_{l-1},b_l} = \sum_{i=b_{l-1}+1}^{b_l-1} \sum_{j>i}^{b_l} s_{\pi(i),\pi(j)}, \tag{18}$$

$$B_{b_{l-1},b_l} = \sum_k c_k \left\lceil \frac{\sum_{i=b_{l-1}+1}^{b_l} t_{\pi(i),k} d_{\pi(i)}}{A_k} \right\rceil. \tag{19}$$

In Eq. (19) the symbol $\lceil x \rceil$ indicates the smallest integer value bigger than x.

Objective function (13) minimizes the weighted sum of the total dissimilarity between the parts and the total machine investment cost. Constraint (14) ensures that each part family must contain at least one part. Constraint (15) makes sure that the cell size limit is not violated. Constraint (16) restricts the number of cells allowed to be formed. Finally, constraint (17) represents that the first cell must be partitioned from the first breaking node.

By using the dynamic programming, the partition problem can be sequentially solved in stages from 1 to c^{max}. Let l be the index of stages, $f_l(b_l, b_{l-1})$ indicates the objective function value at stage l, when at this stage the permutation of parts is partitioned from node $b_{l-1}+1$ to b_l, and $f^*_{l-1}(b_{l-1})$ be the optimum objective function value at stage $l-1$, when its breaking node is b_{l-1}. Therefore, the partition problem at stage l can be stated as follows:

$$f_l(b_l, b_{l-1}) = f^*_{l-1}(b_{l-1}) + w'_1 S_{b_{l-1},b_l} + w'_2 B_{b_{l-1},b_l}. \tag{20}$$

Subject to:

$$\max\{l, P - (c^{max} - l)NP\} \le b_l \le \min\{P, l \times NP\}, \tag{21}$$

$$\max\{l, P - (c^{max} - l)NP, b_l - NP\} \le b_{l-1} \le \min\{P, l \times NP, b_l - 11\}, \tag{22}$$

where $b_0 = 0$, $f^*_0(b_0) = 0$ and $f^*_l(b_l) = \min_{b_{l-1}} f_l(b_l, b_{l-1})$. In addition, constraints (21) and (22) control the cell size limit and the maximum number of cells constraints to avoid infeasible solution.

Note that, the proposed dynamic programming approach partitions a string of solution into exactly C^{max} cells. Therefore, the optimum objective function value, $TC^*(\overline{\pi})$, and the optimum number of cells, $L^*(\overline{\pi})$, for permutation $\overline{\pi}$ are obtained as follows:

$$TC^*(\overline{\pi}) = \min_{\{l|b_l=P\}} \{f^*_l(b_l)\} \text{ and } L^*(\overline{\pi}) = \{l | b_l = P, f^*_l(b_l) = TC^*(\overline{\pi})\}. \tag{23}$$

The pseudo code of the proposed dynamic programming algorithm is given in Fig. 2.

```
Function $TC^*(\overline{\pi})$
  $TC^*(\overline{\pi}) \leftarrow \infty$
  For $l = 1$ to $c^{max}$ do
    For $i = \max\{l, P - (c^{max} - l)NP\}$ to $\min\{P, l \times NP\}$ do
      $f_l^*(i) \leftarrow \infty$
      For $j = \max\{l - 1, P - (c^{max} - l + 1)NP, i - NP\}$ to $\min\{P, (l - 1)NP, i - 1\}$ do
        If $f_l^*(i) > f_{l-1}^*(j) + w_1 S_{j,i} + w_2 B_{j,i}$ then
          $f_l^*(i) \leftarrow f_{l-1}^*(j) + w_1 S_{j,i} + w_2 B_{j,i}$
        End if
      End for
      If $i = P$ and $TC^*(\overline{\pi}) > f_l^*(i)$ then
        $TC^*(\overline{\pi}) \leftarrow f_l^*(i)$
        $L^*(\overline{\pi}) \leftarrow l$
      End if
    End for
  End for
End function
```

Fig. 2. Pseudo code of the proposed dynamic programming algorithm for evaluating the optimal objective function value of a permutation of parts

3.2.2. An illustration

To illustrate the implementation of the dynamic programming algorithm, an example with 10 parts and 10 machines is solved. The input parameters for this example are given in Table 1. For simplicity, the purchase prices of all machines are assumed to be 1$. Also, the design factors are assumed as follows: $NP = 4$, $C^{max} = 4$, $w_1^{'} = 0.2$ and $w_2^{'} = 0.8$. The proposed dynamic programming algorithm is implemented on typical solution $\overline{\pi} = (10, 5, 1, 7, 9, 3, 2, 6, 4, 8)$ to partition it into part families and determine the number of machines in each cell. For this permutation, the increased dissimilarity, $S_{i,j}$, and the increased machine investment cost, $B_{i,j}$ of partitioning a cell from breaking node i to breaking node j were calculated and reported in Table 2.

Table 1
Data of the example used in the illustration of the dynamic programming algorithm

M/P	1	2	3	4	5	6	7	8	9	10	A_k
1	2.95	0	2.2	0	0	0	0	0	0	4.61	230
2	2.76	5.18	1.89	3.89	0	5.14	0	0	0	0	230
3	5.54	4.29	0	0	0	0	0	0	0	0	230
4	2.91	0	0	1.97	2.59	4.01	0	2.7	0	0	230
5	0	0	0	4.28	0	4.51	0	0	0	0	230
6	1.92	0	0	0	0	0	2.23	0	5.52	0	230
7	0	0	0	0	3.4	0	1.16	4.72	0	2.49	230
8	0	5.32	0	0	0	0	0	3.75	3.85	0	230
9	0	0	0	0	0	0	4.04	0	0	1.83	230
d_i	33	30	20	11	18	17	46	46	16	23	

Table 2
Main inputs of dynamic programming calculated for the illustrative example

		$S_{i,j}$										$B_{i,j}$									
$\pi(i)$	i/j	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
10	1	0	0.75	2.44	4.55	—	—	—	—	—	—	3	4	7	7	—	—	—	—	—	—
5	2		0	0.83	2.44	5.02	—	—	—	—	—		2	6	7	9	—	—	—	—	—
1	3			0	0.86	2.44	5.04	—	—	—	—			5	7	9	9	—	—	—	—
7	4				0	0.75	2.75	5.25	—	—	—				3	4	6	7	—	—	—
9	5					0	1	2.5	5.05	—	—					2	4	5	8	—	—
3	6						0	0.75	2.3	3.85	—						2	4	7	7	—
2	7							0	0.8	1.6	4							3	6	6	8
6	8								0	0	1.6								3	3	5
4	9									0	0.8									3	5
8	10										0										3

As the maximum number of cells is equal to 4 cells (i.e., $C^{max} = 4$), the dynamic programming is solved in 4 stages. The summary of the computations are reported in Tables 3–6. According to the results of Tables 5 and 6 we conclude that partitioning permutation $\bar{\pi} = (10, 5, 1, 7, 9, 3, 2, 6, 4, 8)$ into 3 cells results in better objective function value, because $f_3^*(10) < f_4^*(10)$. Therefore, the optimal objective function value, $TC^*(\bar{\pi})$, and the optimal number of cells, $L^*(\bar{\pi})$, are obtained as 15.33 and 3, respectively. The final solution is shown in Table 7.

Table 3
Optimal objective function value at stage $l = 1$

		$f_1(b_1, b_0) = 0.2 \times S_{b_0,b_1} + 0.8 \times B_{b_0,b_1}$		
$\pi(b_1)$	b_1	0	$f_1^*(b_1)$	b_0^*
10	1	2.4	2.4	0
5	2	3.35	3.35	0
1	3	6.088	6.088	0
7	4	6.51	6.51	0

Table 4
Optimal objective function value at stage $l = 2$

		$f_2(b_2, b_1) = f_1^*(b_1) + 0.2 \times S_{b_1,b_2} + 0.8 \times B_{b_1,b_2}$					
$\pi(b_2)$	b_2	1	2	3	4	$f_2^*(b_2)$	b_1^*
5	2	4	—	—	—	4	1
1	3	7.366	7.35	—	—	7.35	2
7	4	8.488	9.122	8.488	—	8.488	1, 3
9	5	10.604	11.038	9.438	8.11	8.11	4
3	6	—	11.558	11.438	9.91	9.91	4
2	7	—	—	12.738	11.01	11.01	4
6	8	—	—	—	13.92	13.92	4

Table 5
Optimal objective function value at stage $l = 3$

		$f_3(b_3, b_2) = f_2^*(b_2) + 0.2 \times S_{b_2,b_3} + 0.8 \times B_{b_2,b_3}$								
$\pi(b_3)$	b_3	2	3	4	5	6	7	8	$f_3^*(b_3)$	b_2^*
3	6	12.208	12.7	11.888	9.71	—	—	—	9.71	5
2	7	—	14	12.988	11.46	12.31	—	—	11.46	5
6	8	—	—	15.898	14.17	14.87	13.41	—	13.41	7
4	9	—	—	—	14.48	15.03	13.41	16.32	13.41	7
8	10	—	—	—	—	17.11	15.33	18.08	15.33*	7

Table 6
Optimal objective function value at stage $l = 4$

		$f_4(b_4, b_3) = f_3^*(b_3) + 0.2 \times S_{b_3,b_4} + 0.8 \times B_{b_3,b_4}$						
$\pi(b_4)$	b_4	6	7	8	9	10	$f_4^*(b_4)$	b_3^*
8	10	16.91	15.78	17.57	15.81	—	15.78	7

Table 7
Optimal partitioning of solution $\bar{\pi} = (10, 5, 1, 7, 9, 3, 2, 6, 4, 8)$

		Number of each machine type in each cell								
	Part families	M1	M2	M3	M4	M5	M6	M7	M8	M9
Cell 1	10, 5, 1, 7	1	1	1	1	0	1	1	0	1
Cell 2	9, 3, 2	1	1	1	0	0	1	0	1	0
Cell 3	6, 4, 8	0	1	0	1	1	0	1	1	0

3.3. Cooling schedule and moving to a neighboring solution

SA algorithm works with a controlled cooling schedule, which is also called the annealing schedule. Starting from initial temperature T_0, the temperature is gradually decreased through a pre-determined cooling schedule. The most commonly used cooling function is the geometric decrement function, $T_t = \alpha \times T_{t-1}$, first proposed by Kirkpatrick et al. (1983). Where α is the cooling rate and takes value between 0 and 1. In this study, this function is used to decrease the temperature. The value of the temperature at the beginning of the schedule should be large enough so that most of the initial movements are accepted. However, as the temperature reduces, the probability of accepting a non-improving solution reduces. To calculate the initial temperature, we modify and combine the approaches proposed by (Tavakkoli-Moghaddam et al., 2008; Ghezavati & Saidi-Mehrabad, 2011). So, the initial temperature can be calculated as follows:

$$T_0 = \frac{\sum_{n=1}^{100} \left| TC^*\left(\overline{\pi}_n^1\right) - TC^*\left(\overline{\pi}_n^2\right) \right|}{-100 \times \ln(0.95)}, \tag{24}$$

where $\overline{\pi}_n^1$ and $\overline{\pi}_n^2$ are two randomly generated solutions at nth trial.

At each iteration (temperature), a generation mechanism called *Move* is applied to transform the current solution into a neighboring (new) solution. Three move operators are proposed, namely *Swap*, *Change* and *Inverse* operators. The *Swap* operator swaps the order of two randomly selected parts. The *Change* operator changes the order of a randomly selected part. The *Inverse* operator reverses the order of parts between two randomly selected points. To illustrate these operators assume that the current solution is $\overline{\pi}^{\text{Current}} = (5,3,6,1,8,2,4,7)$, if the order of part 2 is changed to 3 (*Change* operator), the neighboring solution becomes $\overline{\pi}^{\text{New}} = (5,3,2,6,1,8,4,7)$. If parts 3 and 8 are chosen for swapping (*Swap* operator), the neighboring solution becomes $\overline{\pi}^{\text{New}} = (5,8,6,1,3,2,4,7)$. Lastly, if the orders between parts 1 and 4 are inverted (*Inverse* operator), the neighboring solution becomes $\overline{\pi}^{\text{New}} = (5,3,6,4,2,8,1,7)$. It should be noted that these operators are performed on the current solution independently to derive a neighboring solution.

```
Set ($T_0, I, N^{max}, \alpha$)
$T_t \leftarrow T_0$
$t \leftarrow 1$
generate($\overline{\pi}^{\text{Current}}$)
$\overline{\pi}^{\text{Best}} \leftarrow \overline{\pi}^{\text{Current}}$
Repeat
— $n \leftarrow 0$
— Repeat
—— $m \leftarrow m + 1$
—— Case Random{0, ..., 6} of
——— 0: $\overline{\pi}^{\text{New}} \leftarrow$ Swap($\overline{\pi}^{\text{Current}}$)
——— 1: $\overline{\pi}^{\text{New}} \leftarrow$ Change($\overline{\pi}^{\text{Current}}$)
——— 2: $\overline{\pi}^{\text{New}} \leftarrow$ Inverse($\overline{\pi}^{\text{Current}}$)
——— 3: $\overline{\pi}^{\text{New}} \leftarrow$ Swap and Change($\overline{\pi}^{\text{Current}}$)
——— 4: $\overline{\pi}^{\text{New}} \leftarrow$ Swap and Inverse($\overline{\pi}^{\text{Current}}$)
——— 5: $\overline{\pi}^{\text{New}} \leftarrow$ Change and Inverse($\overline{\pi}^{\text{Current}}$)
——— 6: $\overline{\pi}^{\text{New}} \leftarrow$ Swap, Change and Inverse($\overline{\pi}^{\text{Current}}$)
—— End case
—— $\Delta \leftarrow TC^*(\overline{\pi}^{\text{New}}) - TC^*(\overline{\pi}^{\text{Current}})$
—— If $\Delta < 0$ or $U(0,1) < \exp(-\Delta/T_t)$ then
——— $\overline{\pi}^{\text{Current}} \leftarrow \overline{\pi}^{\text{New}}$
——— $n \leftarrow n + 1$
——— If $TC^*(\overline{\pi}^{\text{New}}) < TC^*(\overline{\pi}^{\text{Best}})$ then
———— $\overline{\pi}^{\text{Best}} \leftarrow \overline{\pi}^{\text{New}}$
——— End if
—— End if
— Until $n = N^{max}$
— $t \leftarrow t + 1$
— $T_t \leftarrow \alpha \times T_t$
Until $t = I$
```
Fig. 3. Pseudo code of the proposed SA algorithm

After generating a neighboring solution, the change in the objective function value is calculated by $\Delta = TC^*(\bar{\pi}^{\text{New}}) - TC^*(\bar{\pi}^{\text{Current}})$. If the change in each transition represents a reduction in the objective function value i.e., $\Delta < 0$, the transition to the new solution is accepted. Otherwise, the non-improving solution is accepted with a specified probability function $\exp(-\Delta/T_t)$. Accepting non-improving solutions enables the SA to avoid the entrapment in local optima. This mechanism at each temperature is repeated until N^{max} accepted transitions are met. Where N^{max} is proportional to the length of string used in the solution encoding (i.e., $N^{max} = N \times P$).

3.4. Stopping criteria

The proposed SA algorithm is terminated when a specified number of iterations, I, is reached. The detailed steps in the implementation of the proposed SA algorithm are given in Fig. 3.

4. Experimental studies

The proposed dynamic programming-enhanced SA algorithm was coded in the Embarcadero Delphi XE programming language and implemented on a Windows 7 PC with 2.4 GHz CPU and 2 GB RAM. As the quality of solution is usually sensitive to the SA parameters, experimental tests are conducted to set the SA parameters. Then, the performance of the SA is evaluated in two sections. In the first section, a set of randomly generated instances are solved by the SA, and the results are compared with the solutions obtained by the B&B method. In the second section, numerical examples adopted from the literature are solved by the SA algorithm and the results are compared with those from the literature.

4.1. Parameter settings for SA

As widely known, settings of SA parameters critically affect the solution efficiency and effectiveness. The parameters of the proposed SA algorithm include: cooling rate (α), number of accepted transitions at each iteration (N^{max}) and number of iterations (I). The value of each parameter is selected from a set of predefined values given in Table 8. It should be noted that these values have been obtained based on our initial computational experience and the result of similar algorithms in the literature. Five test problems with $P = 15, 20, 25, 30, 35$ are used in the parameter settings. The input parameters of these test problems are generated randomly according to Table 9. In this table, the term "U" implies uniform distribution and the term '$\sum_k a_{ik}$' denotes the number of operations of each part which is selected randomly between 3 and 5. All the problems are investigated for $w_1 = w_2 = 0.5$. To calculate w'_1 and w'_2, we first determine TD_{L}, TD_{U}, TI_{L} and TI_{U}. In this way, the SA algorithm is solved two times regarding the following objective functions: $\min \varepsilon \times TD + TI$and $\min TD + \varepsilon \times TI$ (the first objective function gives TD_{U} and TI_{L}, and the second one gives TI_{U} and TD_{L}). We then obtain w'_1 and w'_2 using Eq. (12).

Table 8
Experimental factors and their levels

Parameter	Range
α: Cooling rate	(0.6, 0.65, 0.7, 0.75, 0.8)
N ($N^{max} = N \times P$: No. accepted transitions at each iteration)	(2, 3, 4)
I: No. iterations	(20, 30, 40)

Table 9
Data set generation

M	$\lceil P/2 \rceil + 2$	$\sum_k a_{ik}$	Random$\{3, 4, 5\}$
C^{max}	$\lceil P/5 \rceil$	t_{ik}	$U\sim(0.2, 0.8)$min, $\forall\, a_{ik} = 1$
d_i	$U\sim(10, 20)$	NP	7
A_k	$U\sim(20, 30)$min	w_1	0.5
c_k	$U\sim(500, 1000)$ \$	w_2	0.5

For each combination of parameters, each example was solved 10 times (in total $5 \times 10(5 \times 3^2) = 2250$ experiments were done) and Minitab 16 software was used to analyze the results. The effects of parameters on the solution quality and computation time are plotted in Figs. 3 and 4, respectively. To select the best combination of parameters, we consider the trade-off between the solution quality and computation time. As it can be seen in Figs. 4 and 5, decreasing α from 0.7 to 0.6, increasing I from 30 to 40 and increasing N from 3 to 4 does not improve the solution quality meaningfully, but it significantly increases the computation time. According to these plots, the parameters of the SA algorithm are obtained as follows: $\alpha = 0.7$, $I = 30$ and $N = 3$ (the trade-off points have been highlighted in Figs. 4 and 5).

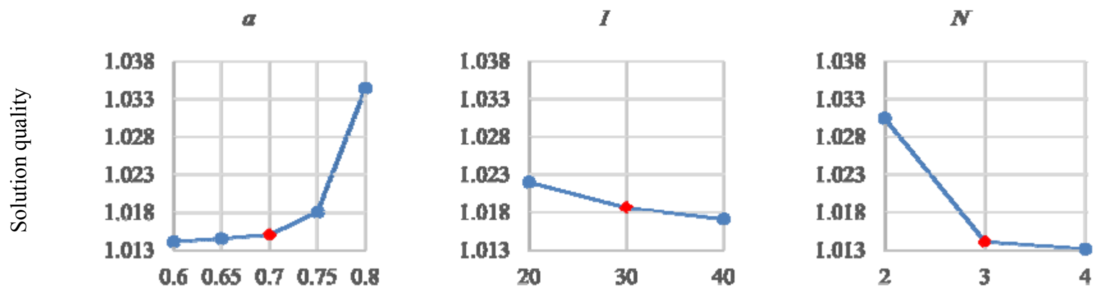


Fig. 4. Effect of parameters on solution quality

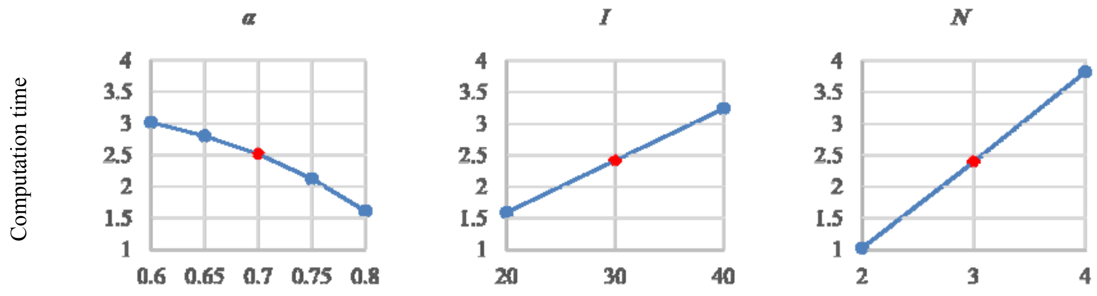


Fig. 5. Effect of parameters on computation time

4.2. Comparison with B&B

Twenty one instances with $P = 10, \dots, 30$ are used to evaluate the performance of the proposed SA algorithm against B&B algorithm, available in the Lingo 11 optimization package. The parameters of these instances are generated randomly according to Table 9. For all the problems we assume that $w_1 = w_2 = 0.5$, and calculate w'_1 and w'_2 as explained in Section 4.1. As it was mentioned before, the CF problems are NP-hard. Hence, some problems may not be solved optimally by the Lingo in an acceptable amount of time. Thus, the solver is interrupted after 7200 seconds (2 hours) and the optimality gap (the relative gap between current feasible solution and best bound on optimal solution) is reported. From the other side, due to the stochastic nature of the SA algorithm, each problem is solved 30 times and the best result is considered for comparison. Table 10 demonstrates the results of comparison. The columns of this table are defined as follows: the best objective function value obtained by the Lingo ($TC_{\text{B\&B}}$), the bound on the objective function value (TC_{Bound}), the time spent on solving the problems by the Lingo ($T_{\text{B\&B}}$), the relative gap between the objective function value and its bound (Opt. Gap), where Opt. Gap $= \left(\frac{TC_{\text{B\&B}} - TC_{\text{Bound}}}{TC_{\text{B\&B}}}\right) \times 100$, the best objective function value found in 30 runs of the SA (TC_{Best}), the mean of CPU times in 30 runs of the SA (T_{SA}), the mean of the objective function values (μ_{TC}), the standard deviation of the objective function values (σ_{TC}), the number of times that the solution of the SA is better than that obtained by the Lingo (R), and the relative gap between TC_{Best} and $TC_{\text{B\&B}}$(Gap), where Gap $= \left(\frac{TC_{\text{B\&B}} - TC_{\text{Best}}}{TC_{\text{B\&B}}}\right) \times 100$.

Table 10

Summary of comparison between the Lingo (B&B) and SA solutions

	B&B (Lingo)				SA					
Problem size ($P \times M \times C^{max}$)	$TC_{\text{B\&B}}$	TC_{Bound}	$T_{\text{B\&B}}$ (s)	Opt. Gap (%)	TC_{Best}	T_{SA}(s)	μ_{TC}	σ_{TC}	R	Gap (%)
10×7×2	5.0637	5.0637	1	0.00	5.0637	0.054	5.0637	0	30	0.00
11×8×3	1.7740	1.7740	20	0.00	1.7740	0.113	1.774	0	30	0.00
12×8×3	1.9420	1.9420	82	0.00	1.9420	0.139	1.942	0	30	0.00
13×9×3	2.6699	2.6699	286	0.00	2.6699	0.183	2.6713	0.0028	27	0.00
14×9×3	2.3722	2.3722	520	0.00	2.3722	0.222	2.3864	0.0193	25	0.00
15×10×3	5.3706	5.3706	1584	0.00	5.3706	0.294	5.4134	0.0698	21	0.00
16×10×4	2.1074	1.6601	>7200	21.23	2.1074	0.417	2.1175	0.0081	26	0.00
17×11×4	2.1741	1.6307	>7200	24.99	2.1475	0.532	2.1709	0.0290	21	1.22
18×11×4	2.0623	1.4838	>7200	28.05	2.0384	0.623	2.0486	0.0164	24	1.16
19×12×4	2.7085	1.9512	>7200	27.96	2.6787	0.733	2.7072	0.0200	21	1.10
20×12×4	2.2648	1.5771	>7200	30.36	2.2180	0.823	2.2408	0.0224	28	2.07
21×13×5	1.8171	1.2110	>7200	33.36	1.7632	1.178	1.7812	0.0156	30	2.97
22×13×5	3.1555	2.1207	>7200	32.79	3.0378	1.177	3.0675	0.0306	30	3.73
23×14×5	3.0831	1.8643	>7200	39.53	2.9511	1.570	2.9747	0.0185	30	4.28
24×14×5	2.9922	1.7297	>7200	42.19	2.8923	1.689	2.9205	0.0247	30	3.34
25×15×5	3.5653	1.9913	>7200	44.15	3.4155	1.915	3.4392	0.0213	30	4.20
26×15×6	2.1712	1.1261	>7200	48.13	1.9639	2.468	1.9844	0.0217	30	9.55
27×16×6	2.0906	1.0108	>7200	51.65	1.9469	2.856	1.9621	0.0185	30	6.87
28×16×6	2.4578	1.2931	>7200	47.39	2.2474	3.039	2.2796	0.0202	30	8.56
29×17×6	2.5932	1.2535	>7200	51.66	2.3738	3.477	2.4126	0.0304	30	8.46
30×17×6	2.8376	1.2527	>7200	55.85	2.6225	3.745	2.6641	0.0217	30	7.58

The problems with $P = 10, 11, \dots, 15$ were solved optimally by the Lingo. However, the remaining problems were not solved optimally in 2 hours. As it can be seen in column "Opt. Gap", by increasing the problem size the optimality gap increases. This implies the complexity of the problem. From the other side, the results show that in all the problems the solution of the SA is better than or at least the same as that obtained by the Lingo. For these problems, the average times that the SA algorithm found better solution than B&B is almost equal to28 out of 30 runs. In addition, the standard deviation of the objective function values are very small and the mean of the objectives function values are very close to the best objective function values. These demonstrate that the SA algorithm is able to consistently produce good solutions. Moreover, the results indicate that the SA algorithm is able to solve the problems in a very short time (less than 4 seconds) compared to the B&B algorithm.

4.3. Comparison with other studies

In this section, twelve problems in different sizes have been selected from the literature. For some problems all of the necessary information was not available in the source papers, so we added the required parameters to the original problems. Table 11 shows the source, size and the parameters added to each problem. For all the problems we assume that $w_1 = w_2 = 0.5$, and calculate w'_1 and w'_2 as explained in Section 4.1. These problems are solved by the proposed SA algorithm and the results are compared with those reported in the literature. As it was mentioned in Section 2, most of studies dealing with the CF problem do not consider machine duplication in the problem formulation, hence to be able to compare the results we have to calculate the number of each machine type in each cell regarding the part family results available in the literature. The summary of comparison is given in Table 12. In this table column "Imp. (%)" shows the improvement percent in the objective function value and is defined as follows: Imp $= \left(\frac{TC_{\text{Lit}} - TC_{\text{SA}}}{TC_{\text{Lit}}}\right) \times 100$; where TC_{SA} is the best objective function value found in 30 runs of the SA algorithm and TC_{Lit} is the objective function value calculated for the solution reported in the literature.

Table 11
Characteristic of the problems selected from the literature

Problem No.	Source	Size $(P \times M)$	d_i	t_{ik}	A_k	c_k
1	Solimanpur et al. (2004)	8×10	$U\sim(10,20)$	$U\sim(0.2,0.8)$	$U\sim(20,30)$	$U\sim(500,1000)$
2	Arıkan and Güngör (2009)	10×9	—	—	—	—
3	Onwubolu and Mutingi (2001)	15×10	$U\sim(10,20)$	$U\sim(0.2,0.8)$	$U\sim(20,30)$	$U\sim(500,1000)$
4	Chang and Lee (2000)	18×4	$U\sim(10,20)$	$U\sim(0.2,0.8)$	50	500
5	Balakrishnan and Jog (1995)	19×12	$U\sim(10,20)$	$U\sim(0.2,0.8)$	$U\sim(20,30)$	$U\sim(500,1000)$
6	Aktürk and Balkose (1996)	20×10	—	—	—	—
7	Adil and Rajamani (2000)	20×20	$U\sim(10,20)$	$U\sim(0.2,0.8)$	$U\sim(20,30)$	$U\sim(500,1000)$
8	Saeedi et al. (2010)	25×15	—	$U\sim(0.2,0.8)$	$U\sim(100,150)$	$U\sim(500,1000)$
9	Filho and Tiberti (2006)	30×15	—	—	1000	$U\sim(500,1000)$
10	Chang and Lee (2000)	30×16	$U\sim(10,20)$	$U\sim(0.2,0.8)$	50	$U\sim(500,1000)$
11	Saeedi et al. (2010)	35×20	$U\sim(10,20)$	$U\sim(0.2,0.8)$	$U\sim(20,30)$	$U\sim(500,1000)$
12	Chan et al. (2006)	43×16	$U\sim(10,20)$	$U\sim(0.2,0.8)$	$U\sim(20,30)$	$U\sim(500,1000)$

The results of Table 12 show that in all the problems (except for problems 3, 5 and 9) both the total dissimilarity, TD_{SA}, and the total machine investment cost, TI_{SA}, obtained by the SA algorithm are better than those obtained for the solution reported in the literature. In addition, it can be seen that in all the problems the objective function value of the SA algorithm, TC_{SA}, is considerably better than that calculated for the solution reported in the literature. For these problems, the average improvement in the objective function value is equal to 13.47%.

Table 12
Result of Comparison for the problems selected from the literature

	Design parameters		Literature				SA					
Problem No.	C^{max}	NP	TD_{Lit}	TI_{Lit}	L_{Lit}	TC_{Lit}	TD_{SA}	TI_{SA}	L_{SA}	TC_{SA}	Time (s)	Imp. (%)
1	4	4	4.133	12486	3	2.367	2.1167	11898	4	1.968	0.04	16.86
2	4	4	9.083	1770340	3	5.015	4.900	1525317	4	4.180	0.09	16.65
3	4	7	4.333	11477	3	3.564	3.333	12327	4	3.498	0.30	1.86
4	5	10	28.000	4000	2	2.453	3.667	4000	4	1.480	0.36	39.67
5	5	10	35.535	27063	3	2.584	35.918	22907	3	2.246	0.65	13.06
6	5	10	35.876	2910	4	1.452	22.287	2448	4	1.140	0.77	21.49
7	5	10	28.610	32682	4	3.005	27.371	30909	4	2.845	1.45	5.34
8	6	10	30.431	33844	4	1.764	28.315	30095	4	1.582	2.20	10.32
9	6	10	14.505	15142	3	1.312	5.600	22908	5	1.295	2.85	1.32
10	6	10	51.237	28596	4	1.842	46.325	25568	5	1.650	3.56	10.39
11	6	10	92.458	38836	4	3.231	61.023	35796	6	2.795	6.99	13.49
12	6	15	108.348	31223	5	2.425	87.783	28131	6	2.153	12.66	11.21

5. Conclusions

In this paper, a SA algorithm was developed to solve a bi-objective CF problem with duplicate machines. In the proposed SA algorithm each solution was represented by a permutation of parts and a dynamic programming algorithm was employed to partition this permutation into part families and determine the number of each machine type in each cell such that the total dissimilarity between the parts and the total machine investment cost are minimized. After setting the SA parameters using statistical experiments, a set of randomly generated instances were used to compare the SA algorithm against the B&B algorithm. The results indicated that the SA algorithm is able to find optimal solution in a very short computational time. In addition, several numerical examples adopted from the literature were solved by the proposed SA algorithm and the results were compared to those from the literature. It was showed that the proposed SA algorithm gives much better results than the others for all problems with an average improvement of 13.47% in the objective function value.

Acknowledgements

This research was supported by the Research Affairs of Kharazmi University under the grant No. 4/581 and this is gratefully acknowledged.

References

Adil, G.K., & Rajamani, D. (2000). The trade-off between intracell and intercell moves in group technology cell formation. *Journal of Manufacturing Systems*, 19, 305–317.

Aktürk, M.S., & Balkose, H.O. (1996). Part-machine grouping using a multi-objective cluster analysis. *International Journal of Production Research*, 34, 2299–2315.

Arıkan, F., & Güngör, Z. (2009). Modeling of a manufacturing cell design problem with fuzzy multi-objective parametric programming. *Mathematical and Computer Modelling*, 50, 407–420.

Arkat, J., Saidi, M., & Abbasi, B. (2007). Applying simulated annealing to cellular manufacturing system design. *International Journal of Advanced Manufacturing Technology*, 32, 531–536.

Balakrishnan, J., & Jog, P.D. (1995). Manufacturing cell formation using similarity coefficients and a parallel genetic TSP algorithm: formulation and comparison. *Mathematical and Computer Modelling*, 21, 61–73.

Banerjee, I., & Das, P. (2012). Group technology based adaptive cell formation using predator-prey genetic algorithm. *Applied Soft Computing*, 12, 559–572.

Boctor, F.F. (1991). A linear formulation of the machine-part cell formation problem. *International Journal of Production Research*, 29, 343–356.

Caux, C., Bruniaux, R., & Pierreval, H. (2000). Cell formation with alternative process plans and machine capacity constraints: A new combined approach. *International Journal of Production Economics*, 64, 279–284.

Chan, F.T.S., Lau, K.W., Chan, P.L.Y., & Choy, K.L. (2006). Two-stage approach for machine-part grouping and cell layout problems. *Robotics and Computer-Integrated Manufacturing*, 22, 217–238.

Chang, P.T., & Lee, E.S. (2000). Multisolution method for cell formation exploring practical alternatives in group technology manufacturing. *Computers and Mathematics with Applications*, 40, 1285–1296.

Chen, W.H., & Srivastava, B. (1994). Simulated annealing procedures for forming machine cells in group technology. *European Journal of Operational Research*, 75, 100–111.

Chiang, C.P., & Lee, S.D. (2004). Joint determination of machine cells and linear intercell layout. *Computers and Operations Research*, 31, 1603–1619.

Filho, E.V.G., & Tiberti, A.J. (2006). A group genetic algorithm for the machine cell formation problem. *International Journal of Production Economics*, 102, 1–21.

Garbie, I.H., Parsaei, H.R., & Leep, H.R. (2008). Machine Cell Formation Based on a New Similarity Coefficient. *Journal of Industrial and Systems Engineering*, 1, 318–344.

Ghezavati, V.R., & Saidi-Mehrabad, M. (2011). An efficient hybrid self-learning method for stochastic cellular manufacturing problem: A queuing-based analysis. *Expert Systems with Applications*, 38, 1326–1335.

Ghosh, T., Sengupta, S., Chattopadhyay, M., & Dan, P.K. (2011). Meta-heuristics in cellular manufacturing: A state-of-the-art review. *International Journal of Industrial Engineering Computations*, 2, 87–122.

Jeon, G., & Leep, H.R. (2006). Forming part families by using genetic algorithm and designing machine cells under demand changes. *Computers and Operations Research*, 33, 263–283.

Kao, Y., & Lin, C.H. (2012). A PSO-based approach to cell formation problems with alternative process routings. *International Journal of Production Research*, 50, 4075–4089.

Kaufmann, L., & Broeckx, F. (1978). An algorithm for the quadratic assignment problem using Benders' decomposition. *European Journal of Operational Research*, 2, 204–211.

Kazerooni, M.L., Luong, H.S., & Abhary, K. (1997). A genetic algorithm based cell design considering alternative routing. *International Journal of Computer Integrated Manufacturing Systems*, 10, 93–107.

Kioon, S.A., Bulgak, A.A., & Bektas, T. (2009). Integrated cellular manufacturing systems design with production planning and dynamic system reconfiguration. *European Journal of Operational Research*, 192, 414–428.

Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P. (1983). Optimisation by simulated annealing. *Science*, 220, 671–680.

McAuley, J. (1972). Machine grouping for efficient production. *The Production Engineer*, 51, 53–57.

Mungwattana, A. (2000). *Design of cellular manufacturing systems for dynamic and uncertain production requirements with presence of routing flexibility*. PhD thesis, Faculty of the Virginia Polytechnic Institute and State University.

Onwubolu, G.C., & Mutingi, M. (2001). A genetic algorithm approach to cellular manufacturing systems. *Computers and Industrial Engineering*, 39, 125–144.

Saeedi, S., Solimanpur, M., Mahdavi, I., & Javadian, N. (2010). Heuristic approaches for cell formation in cellular manufacturing. *Journal of Software Engineering and Applications*, 3, 674–682.

Saghafian, S., & Akbari Jokar, M.R. (2009). Integrative cell formation and layout design in cellular manufacturing systems. *Journal of Industrial and Systems Engineering*, 3, 97–115.

Singh, N. (1993). Design of cellular manufacturing systems: An invited review. *European Journal of Operational Research*, 69, 284–291.

Solimanpur, M., Vrat, P., & Shankar, R. (2004). A multi-objective genetic algorithm approach to the design of cellular manufacturing systems. *International Journal of Production Research*, 42, 1419–1441.

Tavakkoli-Moghaddam, R., Aryanezhad, M.B., Safaei, N. & Azaron, A. (2005). Solving a dynamic cell formation problem using metaheuristics. *Applied Mathematics and Computation*, 170, 761–780.

Tavakkoli-Moghaddam, R., Safaei, N., & Sassani, F. (2008). A new solution for a dynamic cell formation problem with alternative routing and machine costs using simulated annealing. *Journal of the Operational Research Society*, 59, 443–454.

Wemmerlöv, U., & Hyer, N.L. (1989). Cellular manufacturing in the US industry: A survey of users. *International Journal of Production Research*, 27, 1511–1530.

Yin, Y., & Yasuda, K. (2005). Similarity coefficient methods applied to the cell formation problem: a comparative investigation. *Computers and Industrial Engineering*, 48, 471–489.

Yin, Y., & Yasuda, K. (2006). Similarity coefficient methods applied to the cell formation problem: a taxonomy and review. *International Journal of Production Economics*, 101, 329–352.