Annals of Optimization Theory and Practice

Volume 1, Number 3, 15-32
November 2018

DOI: 10.22121/aotp.2019.148030.1015

Local search based meta-heuristic algorithms for optimizing the cyclic flexible manufacturing cell problem

Béla Vizvári[1]✉ • Hüseyin Güden[2] • Mazyar Ghadiri Nejad[3]

[1] Department of Industrial Engineering, Eastern Mediterranean University, Mersin 10, Turkey

[2] Department of Industrial Engineering, Eastern Mediterranean University, Mersin 10, Turkey

[3] Department of Industrial Engineering, Girne American University, Mersin 10, Turkey

✉ vizvaribela@gmail.com

(Received: June 8, 2018/ Accepted: August 3, 2018)

Abstract Flexible robotic cells are used in many real-life industries to produce standardized items at a high production speed. These cells use high-tech and expensive machines. Determining the schedules of these cells for the most efficient usage arises as an important optimization problem in those industries. In this study, the scheduling problem of a flexible robotic cell is considered. Machines are identical and parallel. In the cell, there is an input and an output buffer wherein items being processed and the finished items are kept, respectively. There is a robot performing the loading/unloading operations of the machines and transporting the items. The system repeats a cycle in its run. It is assumed that each machine processes one part in each cycle. The cycle time depends on the order of the loading/unloading activities. Therefore, determining the order of these activities for the minimum cycle time is needed. We propose a new mathematical model to solve the problem. For large size problems, three metaheuristic algorithms based on local search algorithm are proposed. In the metaheuristics, in order to compute the minimum cycle time of a given solution a linear programming model is needed to be solved which is one of the recent cases in the literature to the best of our knowledge. Several numerical examples are solved by the proposed algorithms and their performance and solutions are compared.

Keywords Scheduling; Flexible manufacturing system; Robotic cell; Metaheuristic

1. Introduction

Flexible manufacturing cells containing a number of CNC machines are controlled by an automated control system, and the materials can be handled by robots between the machines Nejad *et al.* (2018b). Such cells are widely used in many real-life industries such as chemical, electroplating and metal cutting industries to have a high standardization and to increase the efficiency.

In this study, there is a cell containing a number of parallel and identical CNC machines located on a line to process identical items. There is also an input buffer at the beginning of the line wherein the items to be processed are kept and an output buffer at the end of the line wherein the finished items are stored. A robot transports the items in the system and loads/unloads the machines, and a computer controls the robot. A system with two machines is illustrated in Figure 1.

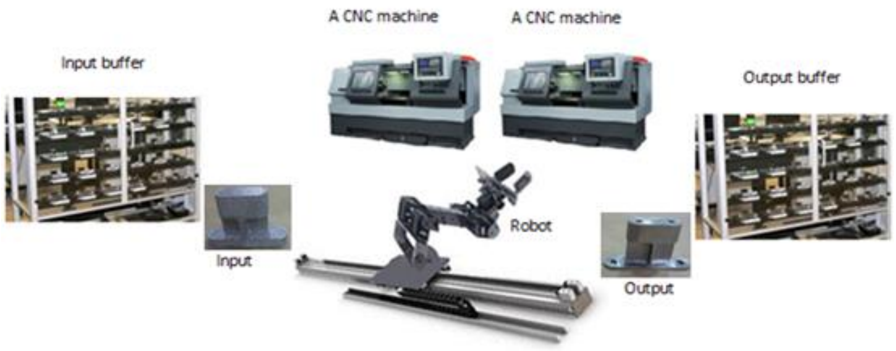

Figure 1. A robotic cell.

Cell manufacturing indicates a connective system among product oriented and process oriented systems. Using a robot in such cells helps the system to produce standardized items at a high production speed Caramia and Mari (2017). The considered system repeats a cycle in its long run. Each machine should be loaded once in a cycle. If the system is in a particular state at the beginning of a cycle, it reaches the same state at the end of the cycle and then repeats the same actions in the same order in the subsequent cycles. The duration of a cycle is called cycle time. Decreasing the cycle time in such a system means increasing the production rate. The cycle time depends on the order of the loading, unloading activities by the robot. In such a system, determining the order of these activities to minimize the cycle time or to maximize the production rate arises as an optimization problem.

Gultekin *et al.* (2009) presented a mathematical model for the problem and expressed that the problem is NP-hard. In the present study, we first propose a new mathematical model and compare it with their model. Then, we present metaheuristic algorithms, such as local search (LS), stochastic local search (SLS) and adaptive local search (ALS), to solve the larger problems. When some metaheuristics are desired to be developed for solving the problem it is noticed that even the order of the activities are known it is not trivial to compute the minimum cycle time. For a given solution, i.e., the order of the activities, in order to compute the minimum cycle time a linear programming model is needed to be solved. The reason for such a need is explained in the following sections.

Finally, we analyze the performances of the proposed metaheuristics using several numerical instances.

This article is organized as follows: in the next section, the related literature is reviewed. Definition and formulation of FMC problems for identical parts and parallel machines are presented in Section 3. Section 4 provides the related information to develop metaheuristic algorithms including representation of the solution, generating initial solution, objective function computations, generating the next solution, stopping criteria and parameter settings. Section 5 includes experimental results of the small- and large-size problems and gap analysis for very large-size problems. Finally, Section 6 contains a discussion and conclusion of the study.

2. Related literature

Being a pioneer research group, Sethi *et al.* (1992) dealt with a robotic cell intending to maximize the long-run average throughput of the system. Their problem was related to producing a single part using two or three machines in a centered-layout robotic cell. Crama (1997) presented a dynamic programming model aimed to produce identical parts by using in-line layout robotic cells. Using this approach, Crama and Van de Klundert (1999) found a shortest cyclic schedule for the robot moves. In addition, they proved that when there are three machines in a cell, the minimum long-run average cycle time is achieved by the one-unit cycle. Brauner and Finke (2001) focused on an m-machine cell, and by employing an algebraic approach, proved that when identical parts are produced, the optimality of a one-unit cycle is valid only for two or three machine-cells but not more. Abdekhodaee *et al.* (2004) considered an FMC with parallel machines. They scheduled two-operation and non-preemptable jobs with equal processing and set up times. Dawande *et al.* (2005) considered the one-unit cycle of a cell with multiple robots including a single and dual gripper and presented a lower bound for its cycle time.

Gultekin *et al.* (2008) suggested a new cycle better than all classical robot move cycles reported in the literature for two-machine cells. Moreover, they showed that a robot-centered layout reduces the cycle time compared to an in-line layout, and found an optimal number of machines to minimize the cycle time of m-machine cells. In another study based on the traveling salesman problem, Gultekin *et al.* (2009) presented a mathematical formulation to determine the minimum cycle time for a parallel machine cell considering a general case. Jolai *et al.* (2012) studied a robotic cell scheduling problem with identical part types when machines are flexible and able to swap. They determined all 1-unit cycle times and proposed a novel cycle for robot movements that dominates all robot move cycles in the literature. Yildiz *et al.* (2012) proposed two pure cycles and showed that these two cycles jointly dominate all other pure cycles for a wide range of the processing times. They also presented the worst case for minimizing the cycle time.

Foumani and Jenab (2012) dealt with one-unit cycles for line layout robotic cells and presented a robot move sequence that minimizes the cycle time. They also presented the optimality regions when all parts meet the first machine twice and determined the optimality conditions for different cycles when each part meets both the machines twice. They carried out the sensitivity analysis for both cases and found the best and the worst cycles mathematically. These two researchers extended this work to m-unit pure cycles when the robot is able to swap Foumani and Jenab (2013). They presented a lower bound and introduced a pure cycle that always dominates the others. Ghadirinejad and Mosallaeipour (2013) considered a flexible robotic cell containing two CNC machines and tried two find the best activity order for the robot to minimize the cycle time when the machines have individual input and output buffers. Kim *et al.* (2013) studied the cyclic scheduling problem in a condition when the robot was enriched with a dual-armed cluster tool. They identified the suitable conditions in that the conventional backward and swap sequences yield minimum cycle time. They also proposed two heuristic scheduling strategies and compared them with the conventional scheduling methods. Jiang *et al.* (2014) applied two heuristics to minimize the makespan of a job scheduling problem. They considered a two-machine system where the machines are parallel and identical and the machines are loaded/unloaded by a server.

Recently, Mosallaeipour *et al.* (2018b) studied on a three-machine cell with a mobile robot for producing carton boxes. They aimed to facilitate the decision-makers to determine and utilize the optimal robot activity order schedule according to the characteristics of their problem. Ghadiri Nejad *et al.* (2018) dealt with the scheduling of a multi-machine robotic cell producing identical parts. Without considering individual buffers for the machines, they tried to maximize the throughput of the system in the long run utilizing a metaheuristic algorithm. Moreover, Nejad *et al.* (2018c) considered a bi-objective scheduling problem of a flexible robotic cell to minimize the cyclic production cost of the cell. They proposed a mathematical model and developed an NSGAII for large-sized problems.

In the recent publications, metaheuristic algorithms have been utilized to solve different type of problems such as solving facility location problems Shavarani *et al.* 2018, the assembly line balancing problems Nejad *et al.* (2018a), the cutting problems Mosallaeipour *et al.* (2018a), the emergency response time problems Ghadiri Nejad and Banar (2018), the cost minimization of renewable energy generation Vatankhah Barenji *et al.* (2018) and etc.

In this study, we consider a flexible manufacturing cell including m parallel machines wherein each of them produces only one part in each cycle. Furthermore, the robot moving time between any two machines is equal and all the loading and unloading times are considered constant. Moreover, we propose three metaheuristic algorithms to solve the problem for the large-size cases.

3. Problem definition and formulation

Let the number of the machines in the considered flexible robotic cell, explained in the introduction, is m. The process time of a part on a machine is p. The loading activity of machine i (Li) consists of picking, transferring, and loading a part from the input buffer to the machine i. The unloading activity of machine i (Ui) includes getting the processed part from machine i, and transferring and putting it into the output buffer. Note that the robot stays beside machine i at the end of Li and it stays beside the output buffer at the end of any unloading activity. A cycle time is the duration time from the starting of the system from a specific state and returning to the same state. Each machine may be loaded or emptied at the beginning of the cycle. During a cycle, each machine must be loaded and unloaded only once. Let L is the set of loading activities, U is the set of unloading activities, and A is the set of all loading and unloading activities.

Let ε be the loading/unloading time of each machine and each buffer, and δ be the robot travel time between the input buffer and the first machine, between two consecutive machines, and between the last machine and the output buffer. dab in the following formula gives the time between the completion time of activity and completion time of activity b when activity b follows activity a. Let's activity b is an unloading activity; if the robot finishes activity and the process on machine j has not been completed then activity b cannot be started. In such a case, the robot must wait until finishing the process on machine j. Note that dab does not contain this uncertain amount of waiting time.

$$d_{ab} = \begin{cases} 2\varepsilon + (i+j)\delta & \text{if } a = L_i \text{ and } b = L_j \\ 2\varepsilon + 2(m+1-j)\delta & \text{if } a = U_i \text{ and } b = U_j \\ 2\varepsilon + (m+1+j)\delta & \text{if } a = U_i \text{ and } b = L_j \\ 2\varepsilon + (|i-j|+m+1-j)\delta & \text{if } a = L_i \text{ and } b = U_j, i \neq j \\ 2\varepsilon + (m+1-i)\delta + p & \text{if } a = L_i \text{ and } b = U_i \end{cases}$$

Since the robot performs the same order of activities in a cycle, to prevent permutation and have a fixed cycle, we consider L1 as the first activity. Thus, the time from L1 to the next L1 is the cycle time (T) and the problem is to determine the order of all loading and unloading activities to minimize T. The decision variables employed in this study are as follows:

T: cycle time

ta: completion time of activity $a \epsilon A$

$$x_{ab} = \begin{cases} 1 & \text{if activity } b \text{ follows activity } a \\ 0 & \text{otherwise} \end{cases}$$

$$z_i = \begin{cases} 1 & \text{if } L_i \text{ precedes } U_i \\ 0 & \text{otherwise} \end{cases}$$

The mathematical model of the problem is the following:

Minimize T (1)

subject to

$$t_{L_j} \geq t_{L_i} + d_{L_iL_j}x_{L_iL_j} - \left(1 - x_{L_iL_j}\right)M \quad \forall\, i,j = 1,\dots,m, i \neq j, j \neq 1 \tag{2}$$

$$t_{L_j} \geq t_{U_i} + d_{U_iL_j}x_{U_iL_j} - \left(1 - x_{U_iL_j}\right)M \quad \forall\, i,j = 1,\dots,m, j \neq 1 \tag{3}$$

$$t_{U_i} \geq t_{L_i} + \ d_{L_iU_i} - (1 - z_i)M \quad \forall\, i = 1,\dots,m \tag{4}$$

$$t_{U_i} \geq (t_{L_i} - T) + d_{L_iU_i} - Mz_i \quad \forall\, i = 1,\dots,m \tag{5}$$

$$t_{U_i} \leq t_{L_i} + Mz_i \quad \forall\, i = 1,\dots,m \tag{6}$$

$$t_{L_i} \leq t_{U_i} + (1 - z_i)M \quad \forall\, i = 1,\dots,m \tag{7}$$

$$t_{U_i} \geq t_l + d_{lU_i}x_{lU_i} - \left(1 - x_{lU_i}\right)M \quad \forall\, l \,\epsilon\, A - U_i \tag{8}$$

$$T \geq t_l + d_{lL_1}x_{lL_1} \quad \forall\, l \,\epsilon\, A - L_1 \tag{9}$$

$$\sum_b x_{ab} = 1 \quad \forall\, a, b \epsilon\, A, a \neq b \tag{10}$$

$$\sum_a x_{ab} = 1 \quad \forall\, a, b \epsilon\, A, a \neq b \tag{11}$$

$$x_{ab} \in \{0,1\} \quad \begin{array}{c}\forall a, b \in A;\ z_i \in \{0,1\}, \forall i;\ t_a \geq 0, \\ \forall\, a \in A\end{array} \tag{12}$$

The objective function aims to minimize the cycle time which is depicted by Equation (1). Constraint (2) considers all cases that the robot loads on two different machines consecutively as LiLj. Constraint (3) is related to the cases where an unloading activity is followed by a loading activity. Constraint (4) is related to the loading activities that are followed by an unloading activity. Equation (5) describes all cases where the loading of a machine was done in the previous cycle. If $z_i = 1$ then the loading of machine i is followed by an unloading of the same machine in the same cycle and $t_{U_i} \geq t_{L_i}$. The opposite must be true as well, i.e., if $z_k = 0$ then $t_{L_k} \geq t_{U_k}$, which is shown by constraints (6) and (7). The next constraint helps to calculate all the completion times of the unloading activities. To calculate the cycle time, it needs to add the moving duration time of the robot from the last position to the input buffer plus the time of getting a part from the input buffer and loading it to machine 1, which is done by considering constraint (9). Constraints (10) and (11) are the classical assignment constraints that guarantee that each activity is performed by the robot. Finally, Equation (12) defines the decision variables. It must be mentioned that to prevent cycle permutation like $L_1L_2U_1U_2$ and $U_2L_1L_2U_1$, it is assumed that the cycle starts when machine 1 is loaded ($t_{L_1} = 0$). M is a large number which is at least as great as the cycle time.

4. Developed metaheuristic algorithms

Since the robotic flexible cell problems belong to NP-hard class of problems, optimizer software like CPLEX can only find the optimal solutions of small-size problems. Therefore, in order to solve large-size problems, the local search algorithm which is one of the most efficient metaheuristic algorithms is used. This algorithm explores the search space to find better solutions without doing a further investigation Sevkli and Aydin (2007).

4.1. Representation

In our study, a solution is presented by an array having 2m elements in which the numbers 1 to m correspond to the loading of the first machine to the mth machine and the numbers m+1 to 2m correspond to the unloading of the first machine to the mth machine, respectively. To prevent permutations, the first element of each array is always 1. For example, a four-machine flexible cell having a cycle $L_1L_3L_4U_2U_3U_1U_4L_2L_1$ is presented in Figure 2.

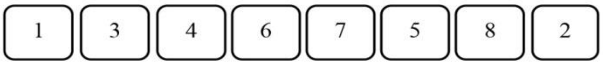

Figure 2. Presentation of a $L_1L_3L_4U_2U_3U_1U_4L_2L_1$cycle for a four-machine flexible cell.

In this array, 1 is the first element and depicts L_1. L_3 and L_4 are shown by 3 and 4 respectively. Since in this example $m = 4$, all unloading activities are shown by 5 to 8; U_2 and U_3 are shown by 6 and 7, respectively. The same procedure is applied to demonstrate the rest of this array.

4.2. Initial solution

An initial solution can be a predefined solution like cycle L1L2…LmU1U2…UmL1 (its array is shown in Figure 3 and discussed in Section 4.1), or it can be based on an iterative construction. For example, after loading the first machine, the activity that has the lowest robot moving time from the previously assigned activity among the remaining activities is selected until the cycle is complete.

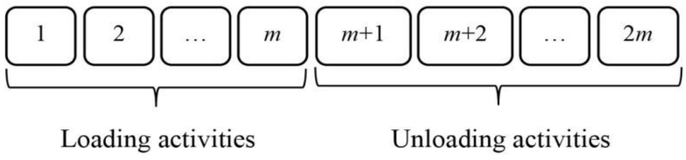

Figure 3. The array of L1L2…LmU1U2…UmL1 cycle.

After some preliminary experiments and testing different initial solutions in practice, to avoid considering a unique initial solution as the starting point of the algorithms, generating a random initial solution was selected. Generating random solutions helps to start from different initial solutions in each run that avoids entrapment in a local optimum.

4.3. Computing cycle time for a given solution

For finding the objective value of each array, all of the parameters, except waiting times, are known. Let's consider the cycle of $L_1L_3L_4U_2U_3U_1L_2U_4L_1$, assuming that loading/unloading times are 1, the robot move time between a pair of two consecutive machines is 2 and all processing times are the same and equal to 80 time units. To calculate the cycle time in this case, the duration time between a set of two activities is calculated by using dab formula in Section 3. For example, the duration time formula for L_1L_3 is $2\varepsilon + 4\delta,$ which is equal to 10 time units. By applying the same procedure, duration times of L_3L_4, L_4U_2, U_2U_3, U_3U_1, U_1L_2, L_2U_4 and U_4L_1 are calculated as 16, 12, 10, 18, 16, 8 and 14, respectively.

To compute waiting times, return time to each machine must be compared with its corresponding processing time. If the processing time is greater than the return time, the waiting time is positive, otherwise, 0. There are two cases that should be considered; if loading of a machine occurs before unloading of the same machine in a cycle, like the machine 1 in this example. In this case, the summation of the duration between a set of two activities from loading a machine to return to the same machine to unload it in addition to the potential waiting times should be considered. For example, the return time to machine 1 is the summation of the times for completing any two consecutive activities from L1 to U3 plus the robot traveling time from U3 to reach beside machine 1, which are $L_1L_3 + L_3L_4 + L_4U_2 + U_2U_3 + 4\delta = 56$, plus potential robot waiting time on machine 2 and machine 3. Since as mentioned in Section 3, before each unloading activity, a positive robot waiting time may exist. Consequently, the robot waiting time on machine 1 is equal to $max\{0, p_1 - (56 + w_2 + w_3)\}$.

On the other hand, if unloading a machine is before loading the same machine in a cycle, like machine 2 in our example, the return time to the machine is the sum of durations from loading that machine to the end of the cycle, i.e., the loading of machine 1, and from loading machine 1 to return to the same machine to unload it in the next cycle. Obviously, potential waiting times should also be considered if there is any. For example, to calculate the return time to machine 2, at first, two cycles can be considered consecutively (see Figure 4). Then, the duration from the first loading of machine 2 to when the robot is ready to unload it, plus w_4 is the return time to machine 2, which is equal to 52+w_4. Therefore, $w_2 = max\{0, p_2 - (52 + w_4)\}$.

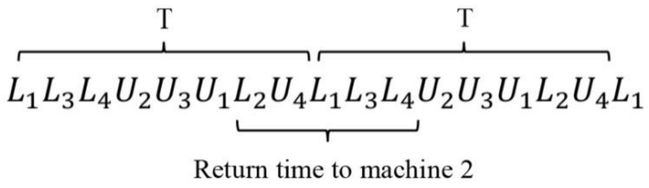

Figure 4. Return time to machine 2 in $L_1L_3L_4U_2U_3U_1L_2U_4L_1$ cycle

Applying the same procedure leads us to reach to the waiting times for machines 3 and 4 equal to $w_3 = max\{0, p_3 - (32 + w_2)\}$ and $w_4 = max\{0, p_4 - (60 + w_1 + w_2 + w_3)\}$ respectively. Therefore, $T = 10 + 16 + 12 + 10 + 18 + 16 + 8 + 14 + w_1 + w_2 + w_3 + w_4 = 104 + w_1 + w_2 + w_3 + w_4$.

Since the aim is to minimize cycle time, the minimal amount of the total waiting times must be computed. On the other hand, increasing in any waiting time will affect other waiting times that are related to. Therefore, a linear programming model should be solved to minimize the cycle time. The linear programming model of our example is as follows:

$$\begin{aligned}
&\mathit{Minimize}\ T = 104 + w_1 + w_2 + w_3 + w_4 \\
&\text{subject to} \\
&w_1 \geq p_1 - (56 + w_2 + w_3) \\
&w_2 \geq p_2 - (52 + w_4) \\
&w_3 \geq p_3 - (32 + w_2) \\
&w_4 \geq p_4 - (60 + w_1 + w_2 + w_3) \\
&w_i \geq 0 \qquad\qquad \forall\, i = 1, 2, 3, 4
\end{aligned}$$

where all the processing times are known and must be substituted before solving the model.

4.4. Generating the next solution

To generate a new solution for the next generation, three different methods based on local search algorithm are considered. The local search algorithm generates neighborhood solutions and tries to find the local optimums, iteratively Mjirda *et al.* (2014). Since this method accepts only a better solution as the current solution, it always modifies the last improved solution. In this study, three different operators that are the shift, swap, and reverse are employed. Figure 5 shows the shift operator for the robot move sequence in a four-machine cell which removes one of the activities from the sequence and moves it to another place in the sequence, randomly.

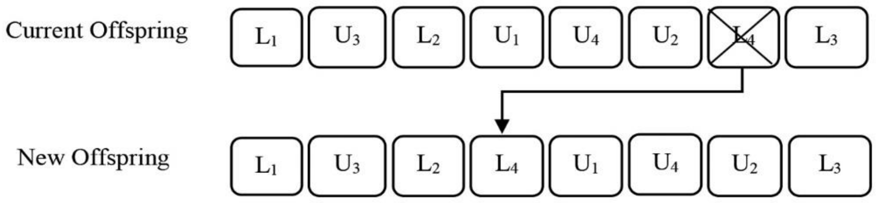
Current Offspring
L_1 U_3 L_2 U_1 U_4 U_2 L_4 L_3
New Offspring
L_1 U_3 L_2 L_4 U_1 U_4 U_2 L_3

Figure 5. Shift operator.

The swap operator finds two activities randomly and switches them together (see Figure 6).

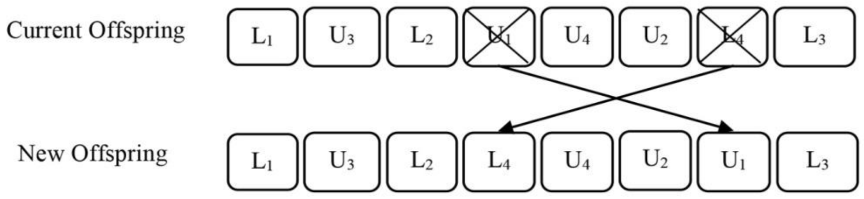

Figure 6. Swap operator.

The reverse operator selects two activities, which are not next to each other, and reverse all the activities among them as it can be seen in Figure 7.

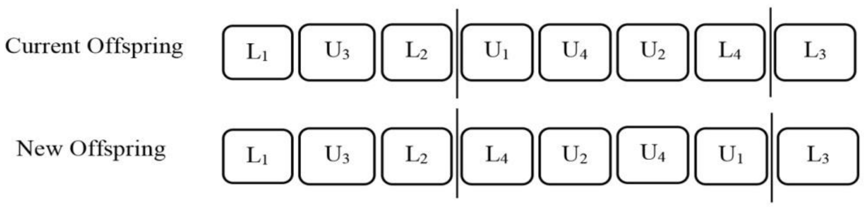

Figure 7. Reverse operator.

In the proposed local search algorithm, shift, swap, and reverse operators are iteratively used in order to find a neighborhood solution. The pseudocode of this method is presented by Algorithm ۱. This algorithm executes Imax iterations (lines 5–15). In each iteration, a new solution is generated by using Shift, Swap, and Reverse operators consecutively; and the solution is compared with the best-known solution generated earlier.

Algorithm 1. Local search (LS)

```
1   Procedure LS ($I_{max}$)
2   OFS $\longleftarrow$ Initial Solution;
3   GB $\longleftarrow$ OFS;
4   $I_{LS} \longleftarrow 0$;
5   While $I_{LS} < I_{max}$ do
6       OFS $\longleftarrow$ Shift (GB);
7       If $T_{OFS} < T_{GB}$ then
8           GB $\longleftarrow$ OFS;
9       OFS $\longleftarrow$ Swap (GB);
10      If $T_{OFS} < T_{GB}$ then
11          GB $\longleftarrow$ OFS;
12      OFS $\longleftarrow$ Reverse (GB);
13      If $T_{OFS} < T_{GB}$ then
14          GB $\longleftarrow$ OFS;
15      $I_{LS} \longleftarrow I_{LS} + 1$;
16  Return $T_{GB}$;
```

To escape from an entrapment in a local optimum, stochastic local search algorithm is employed Ribeiro and Resende (2012). In the present study, this method accepts the worse generated solution in the current iteration with a constant probability (C). Each of the mentioned operators has equal probability to be selected. Algorithm 2 presents the pseudocode of the proposed method. In this algorithm, which executes Imax iterations (lines 6–35), based on a generated random probability (Pr) for each iteration, one of the operators is used (line 7).

Algorithm 2. Stochastic local search (SLS)

```
1   **Procedure** SLS ($I_{max}$)
2   OFS $\longleftarrow$ Initial Solution;
3   GB $\longleftarrow$ OFS;
4   BGS $\longleftarrow$ OFS;
5   $I_{SLS}$ $\longleftarrow$ 0;
6   **While** $I_{SLS} < I_{max}$ **do**
7       Pr $\longleftarrow$ rnd(0,1);
8       **If** Pr < 1/3 **then**
9           OFS $\longleftarrow$ Shift (GB);
10          **If** $T_{OFS} < T_{BGS}$ **then**
11              BGS $\longleftarrow$ OFS;
12          **If** $T_{OFS} < T_{GB}$ **then**
13              GB $\longleftarrow$ OFS;
14          **Else**
15              **If** rnd(0,1) < C **then**
16                  GB $\longleftarrow$ OFS;
17      **elseif** Pr > 2/3 **then**
18          OFS $\longleftarrow$ Swap (GB);
19          **If** $T_{OFS} < T_{BGS}$ **then**
20              BGS $\longleftarrow$ OFS;
21          **If** $T_{OFS} < T_{GB}$ **then**
22              GB $\longleftarrow$ OFS;
23          **Else**
24              **If** rnd(0,1) < C **then**
25                  GB $\longleftarrow$ OFS;
26      **Else**
27          OFS $\longleftarrow$ Reverse (GB);
28          **If** $T_{OFS} < T_{BGS}$ **then**
29              BGS $\longleftarrow$ OFS;
30          **If** $T_{OFS} < T_{GB}$ **then**
31              GB $\longleftarrow$ OFS;
32          **Else**
33              **If** rnd(0,1) < C **then**
34                  GB $\longleftarrow$ OFS;
35      $I_{LS}$ $\longleftarrow$ $I_{LS}$ + 1;
36  **Return** $T_{BGS}$;
```

The other presented metaheuristic algorithm is called adaptive local search. In this method, the probability of selecting each operator is calculated and updated in every fixed number of iterations (n). Equation (13) shows how to calculate the probabilities of each operator.

$$p_y = \frac{x_y}{\sum_1^3 x_y} \qquad \forall\, y = 1,2,3 \qquad (13)$$

where xy is the number of improvements obtained by the operator type y, and 1 to 3 show the shift, swap and reverse operators, respectively. This algorithm is executed by Imax iterations and after every n iterations, the probability of utilizing each operator is calculated and used for the next n iterations. It should be noted that each probability can only be tolerated from 0.1 to 0.8. The pseudocode of this algorithm can be written by applying the mentioned changes in the pseudocode of algorithm 2.

4.5. Stopping criteria

Generally, if a solution found by a metaheuristic algorithm is accurate enough, the iterative method should be stopped. In our study, since the optimal solutions of small-size problems can be found by CPLEX optimizer software, maximum iteration numbers are considered for those problems to find the optimal solutions. For large-size problems, the stopping criteria are finding the lower bound or reaching a predetermined iteration number. The lower bound is computed by solving the problem when the process times are assumed to be zero. In this case, the problem will be a case of the traveling salesman problem.

4.6. The overall method and parameter setting

Some of the parameters must be set beforehand to run the proposed metaheuristic methods. In this research, 1000 iterations for small-size problems and 3000 iterations for large-size problems are considered. To solve each of the algorithms, a new solution is generated; its cycle time is computed and from the next generation, the solution is compared with the best-found solution. Once a solution is selected, it is used to generate a new solution in the next iteration. This procedure is repeated until the algorithm reaches the stopping criteria (as explained in Section 4.5) and the best solution is obtained.

It should be mentioned that after preliminary experiments and using different probabilities of getting the worse generated solution in stochastic local search method, 1% probability is considered. Also, the probabilities of selecting the operators are reset after completing the iterations every 30 times in the adaptive local search method.

5. Experimental Results

Industrial to evaluate the proposed mathematical model and the solution methodologies, several small- and large-size problems are considered and executed on an Intel(R) Pentium(R) Dual CPU E2180 @ 2.00 GHz CPU with 2.0 GB of RAM.

5.1. Exact methods

The proposed mathematical model (TPM) was compared with the only reported model in the literature by Gultekin et al. (TGM) (2009). For this purpose, an FMC with identical parts and processing time of 22, same as in their study, was considered. Both the models have been coded in CPLEX 12.6 software. Table 1 shows the objective functions and the average CPU times of solving the models. Each problem was performed 10 times, except the examples of 6- and 7-machine cell that were executed only once. The CPU times are in second.

Table 1. CPLEX software results for the mathematical models.

No. of machines	Objective function	CPU time	
		TGM	TPM
2	38	0.21	0.20
3	60	0.36	0.29
4	96	3.99	1.85
5	140	341.36	100.28
6	192	-	18338.88

The results (Table 1) demonstrate that TPM is much better than TGM, because of not only TPM has less CPU time than the TGM for solving 2- to 5-machine cell problems, but also TPM can find the optimal solution of 6-machine cell where TGM is not able to solve this problem.

5.2. Metaheuristic algorithms

In this section, the results of solving different problems by local search (LS), stochastic local search (SLS) and adaptive local search (ALS) algorithms are presented. To measure the efficiency of these algorithms, small- and large-size problems are considered. All the metaheuristic algorithms are coded in MATLAB R2013a software. The objective functions (OBF) and the CPU times, shown in the following tables, are the average results of executions carried out 10 times for each problem.

5.2.1. Small-size problems

Since TPM can only solve small-size problems, to evaluate the proposed metaheuristic algorithms, some problems with 4, 5 and 6 machines with different processing times are considered. The processing times of all parts on each machine are considered the same and equal to 20, 80 and 160 (Table 2). It should be noted that when the processing times are equal to 20, the cycle times are equal to the lower bound; and when the processing times are equal to 160, all the cycle times are greater than the lower bound for each problem.

Table 2. Results of the proposed methods for small-size problems.

No. of machines	Process time	Optimal Value	OBF	CPU time (in seconds)			
				TPM	LS	SLS	ALS
4	20	96	96	1.64	0.20	0.46	1.04
	80	108	108	1.59	3.33	4.30	2.32
	160	184	184	1.42	3.18	4.73	2.27
5	20	140	140	101.25	0.43	0.83	0.18
	80	140	140	28.93	3.91	4.33	1.81
	160	188	188	15.75	3.58	6.85	2.66
6	20	192	192	15643.19	0.37	0.33	0.44
	80	192	192	7145.07	2.70	2.53	1.37
	160	198	198	675.75	6.14	9.41	7.43

The results in Table 2 show that all the proposed methods are able to find the optimal solutions for the considered small-size problems. Moreover, by increasing the processing times, the CPU times are increased. Also, when the number of machines in a cell is increased, the CPU time will be increased. Further on, the results show that the ALS algorithm has less CPU times among the other algorithms when the processing time and the number of machines are increased.

5.2.2. Large-size problems

To compare the proposed metaheuristic algorithms precisely, three groups of large-size problems with 10 to 12 machines in a cell are considered (Table 3). Although some methods can find the lower bounds in some of their solutions, this information is hidden because of using the average amount of the results of 10 runs.

Table 3. Results of the proposed metaheuristic algorithms.

No. of machines – LB	Process time	LS		SLS		ALS	
		Cycle time	CPU time	Cycle time	CPU time	Cycle time	CPU time
10 – 480	400	510.8	12.01	504.4	20.60	489.6	25.30
	450	525.2	18.94	526.4	24.87	524.0	17.50
	500	590.0	24.19	588.0	19.49	587.2	16.81
	550	626.8	19.28	617.2	28.71	607.6	24.38
	600	700.4	24.72	686.4	20.21	673.6	13.46
	650	730.4	22.37	728.8	26.42	730.0	16.37
	700	783.2	29.44	767.2	20.47	757.6	17.73
11 – 572	450	580.8	28.68	578.0	16.27	574.4	33.48
	500	598.0	37.57	596.0	25.50	597.6	21.05
	550	638.8	22.03	651.6	29.31	631.6	20.63
	600	711.6	35.70	691.6	30.96	682.0	37.96
	650	739.2	17.49	725.6	28.98	712.4	24.01
	700	793.6	35.75	810.8	24.01	766.8	40.38
	750	822.8	30.90	822.8	30.34	822.8	27.98
12 - 672	500	683.2	20.59	681.6	16.12	675.2	11.87
	550	717.6	36.23	719.2	32.85	695.6	36.21

Table 3. Continued.

	Process time	LS		SLS		ALS	
		Cycle time	CPU time	Cycle time	CPU time	Cycle time	CPU time
	600	729.6	39.85	713.2	31.08	706.0	34.25
	650	739.6	43.92	772.8	11.25	742.0	34.28
	700	806.4	40.23	800.4	33.73	776.4	38.20
	750	865.2	40.57	860.0	28.33	854.0	27.08
	800	906.4	31.95	924.0	33.15	867.2	26.67

The first column of Table 3 shows the number of the machines in a cell and the lower bound of its cycle time. The second column is related to the processing times of the machines. The cycle times and the CPU times of solving the problems by using LS, SLS and ALS methods are shown in the next columns. The results demonstrate that when the processing times are small (less or equal to 500), there is no significant difference among the methods but for larger processing times the ALS method performs much better than the other two methods. Moreover, it is obvious that when the number of the machines or the processing time is increased, the cycle time will also be increased. Additionally, the CPU times of all the methods are similar to each other considering that by increasing the number of machines and their processing times, the CPU times increase slightly.

5.2.3. Gap analysis

To analyze the gaps between solutions of the proposed metaheuristic algorithms and lower bound of each problem, some problems with a very large number of machines in a cell and large processing times are considered. Table 4 shows the results of the gaps and the CPU times of the algorithms for the problems. The gaps are calculated by the formula (14) where LB is the lower bound for each problem and OBF is the objective function of each algorithm for that problem. Gaps are shown in percentage and the remained columns are similar to Table 3.

$$Gap(\%) = \frac{OBF - LB}{LB} * 100 \tag{14}$$

Table 4. Results of the gap analysis.

No. of machines – LB	Process time	LS		SLS		ALS	
		OBF	Gap (%)	OBF	Gap (%)	OBF	Gap (%)
18 – 1440	1100	1488.0	3.33	1459.6	1.36	1452.0	0.83
	1125	1445.8	0.40	1440.8	0.06	1443.2	0.22
	1150	1480.0	2.78	1464.0	1.67	1463.6	1.64
	1175	1480.4	2.81	1503.0	4.38	1472.8	2.28
	1200	1484.0	3.06	1494.4	3.78	1472.4	2.25
19 – 1596	1200	1614.4	1.15	1625.6	1.85	1612.4	1.03
	1225	1616.6	1.29	1632.2	2.27	1618.4	1.40
	1250	1653.2	3.58	1650.4	3.41	1617.6	1.35
	1275	1612.8	1.05	1614.6	1.17	1603.6	0.48

Table 4. Continued.

	Process time	LS		SLS		ALS	
		OBF	Gap (%)	OBF	Gap (%)	OBF	Gap (%)
	1300	1634.4	2.41	1625.4	1.85	1622.8	1.68
20 - 1760	1300	1770.4	0.59	1793.2	1.89	1764.4	0.25
	1325	1807.4	2.69	1791.4	1.78	1762.4	0.14
	1350	1786.8	1.52	1824.4	3.66	1780.8	1.18
	1375	1776.4	0.93	1773.8	0.78	1773.4	0.76
	1400	1790.8	1.75	1773.6	0.77	1771.2	0.64

Table 4, illustrates that the gaps in all three proposed algorithms are between 0.06% to 4.38%. The results illustrate that the gap for ALS is smaller than the other algorithms, and it slightly decreases with increasing the number of the machines.

6. Discussion and conclusion

This paper has presented a novel mathematical model to determine the robot move sequence to minimize the cycle time for the flexible robotic cells problems, wherein, the machines are identical and parallel to each other. In these cells, a robot transports the items from the input buffer, where unproduced items are kept, to the machines and then from the machines to the output buffer, where the finished items are kept, and performs the loading/unloading of the machines. Three metaheuristic algorithms based on local search algorithm are presented to solve large-size flexible robotic cell problems. To evaluate these algorithms, a number of problems have been generated and the solutions have been compared with each other. The results demonstrate that the Adaptive Local Search algorithm outperforms the other proposed algorithms. Developing other metaheuristics for the given problem can be interesting in the further research. Additionally, considering these problems under uncertainty of each parameter can be a future subject. Finally, it is interesting to consider circular layouts. The cases where machines have buffers with limited capacity may also be considered for future research.

References

1. Abdekhodaee, A. H., Wirth, A., & Gan, H. S. (2004). Equal processing and equal setup time cases of scheduling parallel machines with a single server. *Computers & Operations Research*, *31*(11), 1867-1889.
2. Brauner, N., & Finke, G. (2001). Cycles and permutations in robotic cells. *Mathematical and Computer Modelling*, *34*(5-6), 565-591.
3. Crama, Y. (1997). Combinatorial optimization models for production scheduling in automated manufacturing systems. *European Journal of Operational Research*, *99*, 136-153.
4. Crama, Y., & Van de Klundert, J. (1999). Cyclic scheduling in 3-machine robotic flow shops. *Journal of Scheduling*, *2*(1), 35-54.
5. Caramia, M., & Mari, R. (2017). A manufacturing cell formation problem with a maximum cell workload constraint. *IMA Journal of Management Mathematics*, *28*(2), 279-298.

6. Dawande, M., Geismar, H. N., Sethi, S. P., & Sriskandarajah, C. (2005). Sequencing and scheduling in robotic cells: Recent developments. *Journal of Scheduling*, *8*(5), 387-426.
7. Foumani, M., & Jenab, K. (2012). Cycle time analysis in reentrant robotic cells with swap ability. *International Journal of Production Research*, *50*(22), 6372-6387.
8. Foumani, M., & Jenab, K. (2013). Analysis of flexible robotic cells with improved pure cycle. *International Journal of Computer Integrated Manufacturing*, *26*(3), 201-215.
9. Ghadiri Nejad, M., & Banar, M. (2018). Emergency response time minimization by incorporating ground and aerial transportation. *Annals of Optimization Theory and Practice*, *1*(1), 43-57.
10. Ghadiri Nejad, M., Güden, H., Vizvári, B., & Vatankhah Barenji, R. (2018). A mathematical model and simulated annealing algorithm for solving the cyclic scheduling problem of a flexible robotic cell. *Advances in Mechanical Engineering*, 10(1) 1-12.
11. Ghadirinejad, M., & Mosallaeipour, S. (2013). A new approach to optimize a flexible manufacturing cell. In *1st international conference on new directions in business, management, finance and economics* (Vol. 38).
12. Gultekin, H., Akturk, M. S., & Karasan, O. E. (2008). Scheduling in robotic cells: process flexibility and cell layout. *International Journal of Production Research*, *46*(8), 2105-2121.
13. Gultekin, H., Karasan, O. E., & Akturk, M. S. (2009). Pure cycles in flexible robotic cells. *Computers & Operations Research*, *36*(2), 329-343.
14. Jiang, Y., Zhang, Q., Hu, J., Dong, J., & Ji, M. (2015). Single-server parallel-machine scheduling with loading and unloading times. *Journal of Combinatorial Optimization*, *30*(2), 201-213.
15. Jolai, F., Foumani, M., Tavakoli-Moghadam, R., & Fattahi, P. (2012). Cyclic scheduling of a robotic flexible cell with load lock and swap. *Journal of Intelligent Manufacturing*, *23*(5), 1885-1891.
16. Kim, H., Kim, H. J., Lee, J. H., & Lee, T. E. (2013). Scheduling dual-armed cluster tools with cleaning processes. *International Journal of Production Research*, *51*(12), 3671-3687.
17. Mjirda, A., Jarboui, B., Mladenović, J., Wilbaut, C., & Hanafi, S. (2014). A general variable neighbourhood search for the multi-product inventory routing problem. *IMA Journal of Management Mathematics*, *27*(1), 39-54.
18. Mosallaeipour, S., Nazerian, R., & Ghadirinejad, M. (2018a). A Two-Phase Optimization Approach for Reducing the Size of the Cutting Problem in the Box-Production Industry: A Case Study. In *Industrial Engineering in the Industry 4.0 Era* (pp. 63-81). Springer, Cham.
19. Mosallaeipour, S., Nejad, M. G., Shavarani, S. M., & Nazerian, R. (2018b). Mobile robot scheduling for cycle time optimization in flow-shop cells, a case study. *Production Engineering*, *12*(1), 83-94.

20. Nejad, M. G., Kashan, A. H., & Shavarani, S. M. (2018a). A novel competitive hybrid approach based on grouping evolution strategy algorithm for solving U-shaped assembly line balancing problems. *Production Engineering*, 1-12.
21. Nejad, M. G., Kovács, G., Vizvári, B., & Barenji, R. V. (2018b). An optimization model for cyclic scheduling problem in flexible robotic cells. *The International Journal of Advanced Manufacturing Technology*, *95*(9-12), 3863-3873.
22. Nejad, M. G., Shavarani, S. M., Vizvári, B., & Barenji, R. V. (2018c). Trade-off between process scheduling and production cost in cyclic flexible robotic cells. *The International Journal of Advanced Manufacturing Technology*, *96*(1-4), 1081-1091.
23. Ribeiro, C. C., & Resende, M. G. (2012). Path-relinking intensification methods for stochastic local search algorithms. *Journal of heuristics*, *18*(2), 193-214.
24. Sethi, S. P., Sriskandarajah, C., Sorger, G., Blazewicz, J., & Kubiak, W. (1992). Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems*, *4*(3-4), 331-358.
25. Sevkli, M., & Aydin, M. E. (2007). Parallel variable neighbourhood search algorithms for job shop scheduling problems. *IMA Journal of Management Mathematics*, *18*(2), 117-133.
26. Shavarani, S. M., Nejad, M. G., Rismanchian, F., & Izbirak, G. (2018). Application of hierarchical facility location problem for optimization of a drone delivery system: a case study of Amazon prime air in the city of San Francisco. *The International Journal of Advanced Manufacturing Technology*, *95*(9-12), 3141-3153.
27. Vatankhah Barenji, R., Ghadiri Nejad, M., & Asghari, I. (2018). Optimally sized design of a wind/photovoltaic/fuel cell off-grid hybrid energy system by modified-gray wolf optimization algorithm. *Energy & Environment*, 0958305X18768130.