Int. J. Production Economics 135 (2012) 726–740

Contents lists available at SciVerse ScienceDirect

Int. J. Production Economics

journal homepage: www.elsevier.com/locate/ijpe

Multiple part-type scheduling in flexible robotic cells

G. Didem Batur, Oya Ekin Karasan, M. Selim Akturk *

Department of Industrial Engineering, Bilkent University, 06800 Bilkent, Ankara, Turkey

ARTICLE INFO

Article history:
Received 19 April 2010
Accepted 26 September 2011
Available online 29 October 2011

Keywords:
Flexible manufacturing systems
Robotic cell
CNC
Multiple part-type production

ABSTRACT

This paper considers the scheduling problem arising in two-machine manufacturing cells which repeatedly produce a set of multiple part-types, and where transportation of the parts between the machines is performed by a robot. The cycle time of the cell depends on the robot move sequence as well as the processing times of the parts on the machines. For highly flexible CNC machines, the processing times can be adjusted. To this end, this study tries to find the robot move sequence as well as the processing times of the parts on each machine that jointly minimize the cycle time. The problem of determining the best cycle in a 2-machine cell is first modeled as a traveling salesman problem. Then, an efficient 2-stage heuristic algorithm is constructed and compared with the most common heuristic approach of longest processing time (LPT).

1. Introduction

The use of computer controlled machines and automated material handling devices is essential for the required level of automation in manufacturing industries. A manufacturing cell consisting of a number of CNC machines and a material handling robot is called a robotic cell. The implementation of more complex robotic cells necessitates more sophisticated models and algorithms for their optimization. Many studies in the robotic cell literature try to answer this need. A systematic and pioneering study of the problem of finding optimal sequences of parts and robot moves was started by Sethi et al. (1992) with the objective of maximizing the throughput, or in other words minimizing the cycle time. For the problem of 'one part type with two machines' they proved that the optimal solution is a 1-unit cycle. Most of the studies about scheduling in robotic cells assume that each part being processed passes through the same sequence of locations from the input buffer (I), through machines $M_1,\ldots,M_m$ and finally into the output buffer (O) known as a flow-line robotic cell as discussed in Brauner and Finke (2001). Dawande et al. (2005) showed that cyclic schedules which repeat a fixed sequence of robot moves indefinitely are the only ones that need to be considered in order to maximize the long-term average throughput. Hall et al. (1997) considered the scheduling of operations in a manufacturing cell that repetitively produces a family of similar parts on two or three machines served by a robot. For multiple part-type problems in a two-machine cell, they provided an efficient algorithm of complexity $O(n^4)$ (where n defines the number of parts considered) that simultaneously solves the robot move and part sequencing problems.

Robotic cells can process lots that contain different types of parts. Generally, parts of different types have different processing times for a given machine. The term minimal part set (MPS) defines the set of parts containing the same relative proportions of the part types as the relative proportions of the demand. The problem is to find the robot move sequence and the part input sequence for the MPS that jointly minimize the cycle time or the average steady-state cycle time, in a flow-line robotic cell. Sethi et al. (1992) and Kise et al. (1993) considered the multiple part-type problem with the objective of minimizing the makespan and used the known Gilmore and Gomory (1964) algorithm throughout their solution procedures. Aneja and Kamoun (1999) modeled the problem of minimizing the long run average time as a special case of traveling salesman problem (TSP) and provided an algorithm of complexity $O(n \log n)$. Similar applications can be found in scheduling cranes in a shipyard or hoists in electroplating plants. Wen et al. (2010) developed an exact approach that models the problem as a multi-commodity flow problem with side constraints on a network, and also proposed different heuristics to minimize the makespan. Paul et al. (2007) proposed a heuristic approach for a multi-item single-hoist scheduling problem where each operation has a given time window. The literature on the parallel identical machines with a common server is also related to our study. Setup operations considered in these studies are similar to the robot operations in our study. Abdekhodaee et al. (2006) and Hall et al. (2000) considered parallel machines for which each job requires a setup to be carried out, immediately prior to its processing, by a single server, with the processing executed unattended. They provide heuristic algorithms for different cases of the problem.

* Corresponding author. Tel.: +90 312 290 1360; fax: +90 312 266 4054.
E-mail address: akturk@bilkent.edu.tr (M.S. Akturk).

0925-5273/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.ijpe.2011.10.006

CNC machines possess operational flexibility and process flexibility by definition. Akturk et al. (2005) defined process flexibility as the ability to handle a mixture of operations. On the other hand, operational flexibility is defined as the ability to interchange the ordering of several operations for each part type. These two types of flexibilities enable the allocation of any operation to any one of the two machines. As the allocations of the operations change, the processing times on the machines also change accordingly. Akturk et al. (2005) considered the two machine, identical parts robotic cell scheduling problem with operational flexibility. With this definition of the problem, each part has a number of operations to be processed and the problem is to allocate these operations to the machines and is to find the corresponding robot move cycle that jointly minimize the cycle time. The main result of the paper is that the optimal robot move cycle is no longer necessarily a 1-unit cycle as in the setting of Sethi et al. (1992), but a 2-unit robot move cycle may also be optimal for some parameter inputs. They also provided the regions of optimality for each robot move cycle.

Throughout this study we focus on flexible robotic manufacturing cells consisting of CNC machines and producing multiple part types. We consider an in-line robotic cell with no buffers, consisting of two identical machines which are capable of processing all the parts. Two primary problems need to be solved, namely, the scheduling of parts and the sequencing of robot moves for robotic cells. An important difference of this study from the existing literature is that we do not assume the processing times on each machine to be constant. Thus, allocation becomes our third problem to be solved. Throughout the solution procedure, we focus on cyclic schedules trying to minimize the average steady-state cycle time.

In the following section, the notation and basic assumptions pertinent to this study will be introduced. Section 3 presents the proposed mathematical model. In Section 4, the heuristic algorithms developed to solve the problem under study will be detailed. In Section 5 the proposed algorithms will be compared with each other as well as with the classical longest processing time (LPT) algorithm which is well respected in the existing robotic cell scheduling literature. Section 6 summarizes the contributions and provides the concluding remarks of this study.

2. Notation and assumptions

In multiple part scheduling problem, the classical approach is to focus on cycles which produce the minimal part set (MPS) repetitively. In general, the cell processes k different part-types. In one MPS, r_i parts of type i are produced, where $i=1,\ldots,k$. The total number of completed parts in a cycle is $n=r_1+\cdots+r_k$. An *MPS cycle* is a cycle during which the MPS parts enter the system at input, get processed, leave the system at output, and the system returns to the same initial state. Each part is assumed to have a known processing time. By taking the advantage of the flexibility property, we may choose either to perform all the processing of a part completely on any one of the machines or to share the total time among the two machines. Finding the MPS cycle with the minimum cycle time involves the joint consideration of the following decisions: choosing a robot move sequence, determining a part sequence, and allocating the processes on machines.

There are some assumptions of our study that are also common in the literature. We assume all data to be deterministic. Parts are always available at the input buffer and there is always an empty place at the output buffer. We allow no buffer storage to exist between the machines; thus, each part is either on a machine or is being handled by the robot. Neither the robot nor the machines can be in possession of more than one part at any time. The robot and the CNC machines never experience breakdown and never require maintenance. Setup times are assumed to be negligible. No preemption is allowed in the processing of any part. The total processing time is composed of unit times.

Throughout this study each part in the MPS is treated independently since two identical parts belonging to the same part-type might have different allocations. The following parameters will be used:

- n total number of parts to be produced in the MPS.
- P_j processing times of part-types to be produced, $j=1,\ldots,n$.
- ϵ load/unload times of machines by the robot. Consistent with the literature we assume that loading/unloading times for all machines are the same.
- δ time taken by the robot to travel between two consecutive machines. The robot travel time is assumed to be additive. That is, traveling from machine s to machine m is equal to $|s-m|\delta$. We take I, the input buffer, to be machine M_0 and O, the output buffer, to be machine M_3.

Against this background, the objective is to minimize the long run average cycle time required for the repetitive production of one or more minimal part sets. While solving this problem, we determine the allocated processing times of parts on the machines together with the waiting and blocked times of machines for the parts. The accompanying notation will be formally defined in the forthcoming sections.

Allocation and flexibility concepts are currently recognized in the literature only for the single part type production. The following example is aimed to shed light on our assumptions of process and operational flexibility for the case of multiple part-type production.

Example 1. Assume that we have three parts to be completed with corresponding processing times: $P_1=87$, $P_2=84$, and $P_3=57$. ϵ and δ are 1 and 2 time units, respectively. If we assume that all the processes of a job need to be completed on any one of the two machines as in the case of a parallel machines system, the sequence given by the Gantt-chart in Fig. 1 arises as the optimal one with value 173. Note that the optimality of this solution under this restrictive assumption can be justified by the mathematical formulation that will appear in the next section. In Fig. 1, R represents the movements of the robot and the labels 1 and 2 are used to define the first and second machines, respectively. In the initial state of the system, machine 1 is empty and machine 2 is already loaded with part 2.

One potential allocation scenario is to share the processing time of only one of the jobs. Fig. 2 depicts the solution when the first part has 57 units of its processing time allocated to the first machine and 30 units of it allocated to the second machine. Comparing Figs. 1 and 2, one can see that the blocked time that occurred for the first machine (which is equal to the waiting time for the robot) in the first case has been avoided by this arrangement. As will be explained in Section 4 our solution procedure has been constructed in a way to manage these blocked times. The robot movements observed for the allocation scenario are also detailed on a different example in Appendix B for the interested reader. The result for this example is a smaller cycle time value of 142 units, an improvement of 17% over the no allocation case.

When allocation is performed, even though the robot might have to make extra travel and load/unload movements, since the waiting times decrease and the machine capacities are used more effectively, a smaller cycle time could be realized as observed in Example 1 by changing processing time allocations

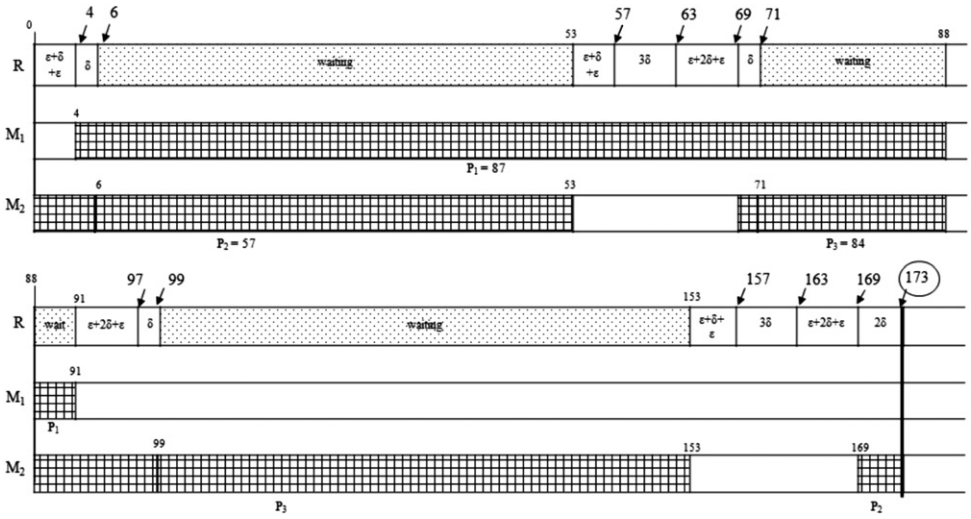

Fig. 1. Example 1 without allocation.

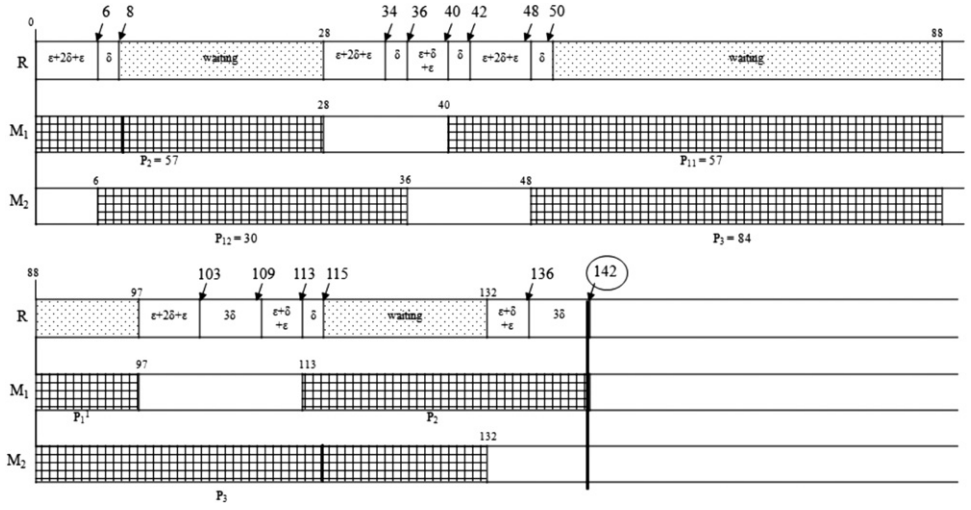

Fig. 2. Example 1 with allocation.

for one part; without decreasing the processing times of parts, without changing the machining conditions, and without using a faster robot.

3. Mathematical model

Our aim in this study is to find a solution that gives both the processing times on machines and the order that these parts are going to be processed. The solution that we are looking for should define the movements of the robot exactly; giving the part to be carried/loaded/unloaded together with the associated machine. This problem can be modeled as a TSP in which the distance matrix consists of decision variables as well as parameters. A basic definition used in this formulation is the following one:

Definition 1. Node i_k identifies the epoch that part i is on station k. The input buffer is denoted as station 1, first and second machines are denoted as stations 2 and 3 respectively, and the output buffer is denoted as station 4. During an MPS cycle, the machines may need to be visited twice, one for loading and one for unloading of the same part since the robot may perform some other activities rather than to wait in front of the machine during the time that the part is being processed. Therefore, stations 5 and 6 are also created as the copies of stations 2 and 3, respectively, in order to account for any potential cyclic solution.

Movements of the robot can be defined as traversals between the nodes. Two types of movements are possible; in the first one a part is carried from a station to another, whereas in the second one, the robot leaves a part on a station and goes to the other one to pick up a new part.

For the formulation, we have an arc set A and a node set N, which are represented as follows:

$N = \{i_k : k = 1,\ldots,6,\ i = 1,\ldots,n\}$,

$A = \{(i_k, j_l) : i_k, j_l \in N$ and the movement from node i_k, where part i is at station k, to node j_l, where part j is at station l, is possible$\}$.

In order to formulate the problem as a TSP, in addition to the ones given in Section 2, we shall need the following parameters and variables:

Parameters:

$Cost_{i_k j_l}$ total time needed for the movement prescribed by the robot activity from where part i is at station k to where part j is at station l.

$Wait_{i_k j_l}$ 1, if there exists a potential waiting time for the movement of the robot from node i_k to node j_l; and 0, otherwise.

Decision variables:

$y_{i_k j_l}$ 1, if robot goes from node i_k to node j_l; and 0, otherwise.
P_{i_k} processing time of part i on station k, $k=2,3,5,6$.
$start_{i_k}$ time when processing of part i starts on station k.
$comp_{i_k}$ time when robot completes the activity related to node i_k.
w_{i_k} robot waiting time for part i in front of the station k.
z_{i_k} 1, if start time of processing of part i on station k is considered before its completion time within a cycle; and 0, otherwise.
$m2_{i_k}$ the part loaded on machine 2 when robot is at node i_k.
$m3_{i_k}$ the part loaded on machine 3 when robot is at node i_k.
C cycle time value.

Variables $m2_{i_k}$ and $m3_{i_k}$ will take values from the set $\{0,1,\ldots,n\}$. When the variable equals to 0, it means that the machine is empty at that moment; whereas its equivalence to any value between 1 and n means that the machine is loaded with that particular part at node i_k.

We need to force the feasibility conditions which indicate that a machine that is already loaded cannot be loaded again and a machine that is already empty cannot be unloaded. Some movements are unreasonable; for example, a part cannot be taken from the input buffer and left to the output buffer without any processing, cannot be taken from the output buffer, or cannot be left on the input buffer. All the possible movements are determined due to these situations and some movements are forbidden within the cycle.

$Cost_{i_k j_l}$ values represent the total time spent in going from node i_k to node j_l. For example, a movement from node i_1 to node i_2 corresponds to the situation that the robot takes a part from the input buffer (ϵ), carries it to the second station (the first machine) (δ), and loads the part (ϵ); makes a total cost of $2\epsilon+\delta$. For our problem, related $Cost_{i_k j_l}$ values are shown in Table 1. The first set of four columns corresponds to the states on which $i=j$ and the second set corresponds to the states on which $i \neq j$. Moreover, the movements with costs included by X's cannot be performed.

Waiting time is one of the main ingredients in the cycle time calculation. It occurs when the robot is ready to unload but when the processing of the loaded part has not been completed yet. It is represented as follows:

$$w_{j_k} = \max\{0, P_{j_k} - v_{j_k}\}, \quad j=1,\ldots,n, \ k=1,\ldots,6, \tag{1}$$

where j is the part loaded on station k and v_{jk} is the total activity time of the robot in between just after loading the machine corresponding to station k by part j and arriving in front of the same machine to unload it.

$Wait_{i_k j_l}$ values are used to see for which movements the robot may need to wait for part i in front of station k. Clearly, for the movements of different parts (i.e. $i \neq j$) no waiting time will be observed and waiting is possible for part i among the movements for which the corresponding costs are underlined in Table 1; so $Wait_{i_k j_l}$ equals to 1 for these movements. For example, when we choose to go from node i_2 to node i_3, we will take part i from machine 2 and load it to machine 3, for which we may need to wait until the processing of the part on machine 2 is completed.

Table 1
Costs of movements performed.

	(j_1)	$(j_2)/(j_5)$	$(j_3)/(j_6)$	(j_4)	(j_1)	$(j_2)/(j_5)$	$(j_3)/(j_6)$	(j_4)
(i_1)	X	$2\epsilon+\delta$	$2\epsilon+2\delta$	X	X	X	X	X
$(i_2)/(i_5)$	X	X	$\underline{2\epsilon+\delta}$	$\underline{2\epsilon+2\delta}$	δ	X	δ	X
$(i_3)/(i_6)$	X	$\underline{2\epsilon+\delta}$	X	$\underline{2\epsilon+\delta}$	2δ	δ	X	X
(i_4)	3δ	X	X	X	3δ	2δ	δ	X

After defining the parameters and variables of the model, we are now ready to proceed with our model. We will start by forcing the y variables to take on proper values. We need to ensure that when there is an incoming arc to a node, there must also be an outgoing arc from it. This fact is guaranteed by the following equation:

$$\sum_{j_l:(j_l,i_k)\in A} y_{j_l i_k} = \sum_{j_l:(i_k,j_l)\in A} y_{i_k j_l}, \quad \forall i_k \in N. \tag{2}$$

In our model, we allow some nodes not to be visited. Thus, the assignment constraints of a TSP should be adapted as follows:

$$\sum_{j_l:(j_l,i_k)\in A} y_{j_l i_k} \leq 1, \ \forall i_k \in N, \quad \sum_{i_k:(j_l,i_k)\in A} y_{j_l i_k} \leq 1, \ \forall j_l \in N. \tag{3}$$

All the parts need to be processed in the system. This requires all the parts to be taken from the input buffer and left to the output buffer exactly once as is defined by Eq. (4)

$$\sum_{j_l:(i_1,j_l)\in A} y_{i_1 j_l} = 1, \ \forall i, \quad \text{and} \quad \sum_{j_l:(j_l,i_4)\in A} y_{j_l i_4} = 1, \ \forall i. \tag{4}$$

Without loss of generality, the system is assumed to start when the robot is in front of the input buffer and ready to take part 1, i.e., at node 1_1.

We use the following Miller–Tucker–Zemlin type constraints in order to calculate the completion times of nodes (Miller et al., 1960)

$$comp_{j_l} \geq comp_{i_k} + cost_{i_k j_l} \cdot y_{i_k j_l} - M(1-y_{i_k j_l}) + Wait_{i_k j_l} \cdot w_{i_k} \quad \forall (i_k, j_l) \in A : j_l \neq 1_1. \tag{5}$$

This equation ensures that when an arc from node i_k to node j_l is used, the completion time of node j_l is at least the sum of the completion time of node i_k, the cost, and the waiting time values corresponding to this movement. When such a movement is not part of the solution, the constraint simply becomes redundant with a large enough chosen big M value.

The following, processing time related equations enable the completion of all the processing. Moreover, they force that the processing times at stations and their duplicates are identical

$$P_{i_2} + P_{i_3} = P_i, \quad \text{where } P_{i_2} = P_{i_5} \quad \text{and} \quad P_{i_3} = P_{i_6}, \forall i. \tag{6}$$

The following constraints establish the relation between the machines and the processing times:

$$P_{i_k} \leq P_i \sum_{j_l:(i_k,j_l)\in A \text{ or } (i_{k'},j_l)\in A} (y_{i_k j_l} + y_{i_{k'} j_l}), \quad i_k \in N \text{ s.t. } k=\{2,3\}. \tag{7}$$

In particular, when node i_k is visited, a processing with a value of at most the total processing time of part i can be performed on station k. Moreover, if station k or its duplicate station k' is not visited by part i, the right hand side of constraint 7 will be zero and no processing will be performed on station k.

We define 'start' and 'comp' variables as the beginning of the processing on a station and the time that the related movement is performed on a node, respectively. The beginning time of a part equals to the time that it is loaded on the related station. This value can be represented by either $start_{i_k}$ or $start_{i_{k'}}$ for part i according to the choice of whether it is loaded on station k or on its duplicate station k', respectively. Since these two stations are the copies of each other, we explicitly force that $start_{i_k} = start_{i_{k'}} \forall i_k \in N$ s.t. $k \in \{2,3\}$

$$start_{i_k} \geq comp_{i_l} - M(1-y_{i_l,i_k}), \quad \forall (i_l,i_k) \in A : k \in \{2,3\}, \ l \neq 4, l \neq k, l \neq k',$$

$$start_{i_k} \geq comp_{i_l} - M(1-y_{i_l,i_{k'}}), \quad \forall (i_l,i_k') \in A : k \in \{2,3\}, l \neq 4, l \neq k, l \neq k'. \tag{8}$$

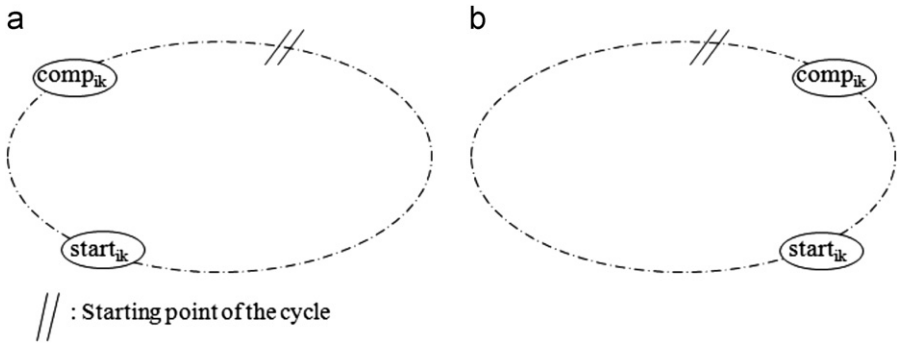

Fig. 3. Choices on load/unload sequence.

Constraint set (8) shows that if there is an arc from node i_l to node i_k or i_k', meaning that if part i is carried from station l to k or k'; processing of part i on machine corresponding to station k or k' starts at the time when the part is left on the machine which is equal to the time when the movement is performed. Since we cannot take any part from the output buffer, l cannot be equal to station 4.

In order to calculate waiting times properly, we resort to the use of another variable associated to each node, namely z_{i_k}, in order to distinguish whether the loading or unloading of a part is performed sooner in a given cycle. Fig. 3 represents the two possible situations. Fig. 3a corresponds to the case when the system starts with part i not loaded on any machine whereas in Fig. 3b station k is already loaded by part i.

We force the depicted relationships by the following constraints:

$$comp_{i_k} \geq start_{i_k} + P_i - M(1-z_{i_k}), \quad \forall i_k \in N,$$

$$start_{i_k} \geq comp_{i_k} + 4\epsilon + 6\delta - Mz_{i_k}, \quad \forall i_k \in N. \tag{9}$$

In these constraints $z_{i_k}=1$ corresponds to $comp_{i_k} \geq start_{i_k} + P_i$, which is the case depicted in Fig. 3a, whereas $z_{i_k}=0$ corresponds to $start_{i_k} \geq comp_{i_k} + 4\epsilon + 6\delta$, which is represented in Fig. 3b. In this case, an already loaded part is to be unloaded and then loaded again. At least a time value of $4\epsilon+6\delta$ which represents the total robot movement between the unload and load of a machine has to realize. For both constraints (8) and (9), M is a big enough number to properly make the constraints redundant whenever needed.

Waiting times realize when a part's processing is not completed by the time the robot arrives to unload it. They are properly assigned by the following constraints:

$$w_{i_k} \geq P_{i_k} - (comp_{i_k} - start_{i_k}) - M(1-z_{i_k}), \quad \forall i_k \in N,$$

$$w_{i_k} \geq P_{i_k} - (C - start_{i_k} + comp_{i_k}) - Mz_{i_k}, \quad \forall i_k \in N. \tag{10}$$

In particular, waiting time will be equal to $\max\{0, P_{i_k} - (comp_{i_k} - start_{i_k})\}$ if $z_{i_k}=1$ and to $\max\{0, P_{i_k} - (C - start_{i_k} + comp_{i_k})\}$ otherwise. These results can also be traced from Fig. 3. For example, in the second case, the total time passage from $start_{i_k}$ to $comp_{i_k}$ equals to the sum of $C - start_{i_k}$ and $comp_{i_k}$. Therefore, we compare this value with the total processing time.

Another set of constraints is constructed in order to define the relation between the movements and the parts loaded on the stations. Considering the movement (i_k, j_l), there are two cases that can be observed, namely:

*Case*1. Robot activity is related to different parts, i.e. $i \neq j$.

This situation refers to the activity that the robot loads part i on station k and travels to station l for part j. Such a movement does not cause any part carriage. Therefore, no difference in terms of the loaded parts is observed for the machines.

*Case*2. Robot activity is performed for the same part, i.e. $i=j$.

This situation refers to the activity that the robot takes a part from station k and travels to station l to load it. In order for such a movement to exist, the following conditions should be satisfied:

- The machine corresponding to station l, which is the one to be loaded, needs to be empty at node i_k, for $l \neq 4$ since there are always an empty space in the output buffer by assumption.
- The machine corresponding to station k which is the one to be unloaded becomes empty at node j_l, for $k \neq 1$ since there is always some parts in the input buffer by assumption.
- This type of a movement results with no difference in terms of the loaded parts for the machines corresponding to the stations other than k, k', l, and l'.

Since stations 1 and 4 correspond to input and output buffers, respectively; we check these relations only for stations 2, 5 and 3, 6 which correspond to the first and second machines, respectively. Now we are ready to list the constraints that are used to force these cases:

*Case*1. If a movement from node i_k to j_l exists and such a movement does not cause any difference in terms of the loaded parts on machines, then $m2_{i_k} = m2_{j_l}$ and $m3_{i_k} = m3_{j_l}$ should be satisfied. These results are obtained by the following constraints:

$$\begin{aligned} -m2_{i_k} + m2_{j_l} &\leq n(1-y_{i_k j_l}), \\ m2_{i_k} - m2_{j_l} &\leq n(1-y_{i_k j_l}), \end{aligned} \tag{11}$$

$$\begin{aligned} -m3_{i_k} + m3_{j_l} &\leq n(1-y_{i_k j_l}), \\ m3_{i_k} - m3_{j_l} &\leq n(1-y_{i_k j_l}). \end{aligned} \tag{12}$$

*Case*2. If a part (part i) is carried from station k to station l;

- Machine corresponding to the station l needs to be empty at the previous station k. Moreover, the machine corresponding to the station k will become empty at the station l. Such requirements are forced via the following constraints:

$$\begin{aligned} &m2_{i_k} \leq n(1-y_{i_k,i_l}) \quad \text{and} \quad m3_{i_l} \leq n(1-y_{i_k,i_l}), \\ &\text{if } k \in \{3,6\} \quad \text{and} \quad l \in \{2,5\}, \end{aligned} \tag{13}$$

$$\begin{aligned} &m3_{i_k} \leq n(1-y_{i_k,i_l}) \quad \text{and} \quad m2_{i_l} \leq n(1-y_{i_k,i_l}), \\ &\text{if } k \in \{2,5\} \quad \text{and} \quad l \in \{3,6\}. \end{aligned} \tag{14}$$

Finally, the cycle time value C should be as large as the completion time of the last node, i.e.

$$C \geq comp_{i_k} + y_{i_k,1_1} \cdot cost_{i_k,1_1} \quad \forall (i_k, 1_1) \in A. \tag{15}$$

Against all this, our optimization problem has the following integer linear programming formulation:

$$\begin{aligned} &\min \quad C \\ &\text{Subject to} \quad (2)-(15) \quad \text{all variables integers.} \end{aligned}$$

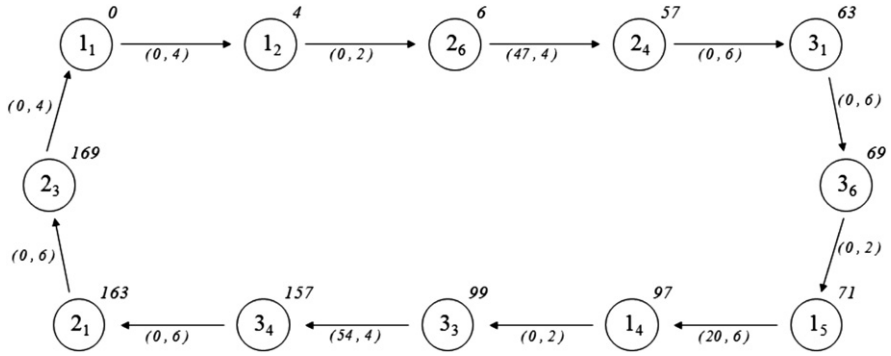

Fig. 4. TSP representation of Example 1 without allocation.

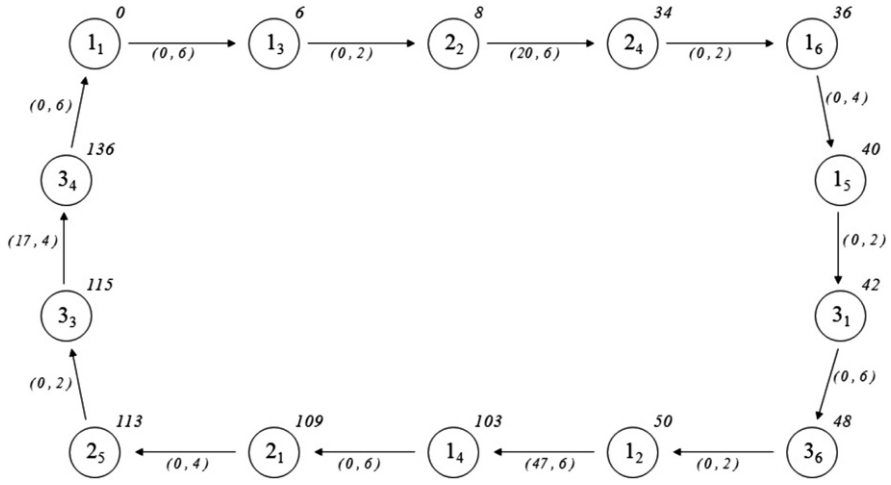

Fig. 5. TSP representation of Example 1 with allocation.

At this point, we would like to remark on some properties of this model. In its current form, the above formulation allows allocation. We could simply forbid the transportation of the parts between the machines by the inclusion of the following constraint set in the above model:

$$y_{i_2,i_3}+y_{i_2,i_6}+y_{i_3,i_2}+y_{i_3,i_5}=0, \quad \forall i. \tag{16}$$

Furthermore, using the same logic and methodology, we can easily adapt this model to the m-machine case.

The above formulation is solved with the data of Example 1. The optimal solutions without allocation and with allocation are given by Figs. 4 and 5, respectively. One may follow these tours with their accompanying Gantt-charts in Figs. 1 and 2, respectively. Node 1_1 is the starting point for both of the examples. The values written on the arcs represent the waiting and travel time values between the nodes, respectively. The optimal y values carry all the necessary information pertaining to the assignments and sequences. In particular, in Fig. 4, the system starts when the first machine is empty and the second one is loaded by part 2. At time 4, the robot loads the first machine by part 1 and goes to the second machine at time 6. As can be seen from Fig. 1, the processing of part 2 is completed at this moment and the robot waits until time 53. At time 57, the part is unloaded and the robot goes to the input buffer at time 63. Part 3 is taken from the input buffer and loaded on the second machine at time 69. The robot now goes to the first machine to unload part 1 and waits until time 91, until the processing is completed. Further steps can also be observed in the same manner.

Comparing Figs. 4 and 5, we can see that some additional nodes are included in the latter one necessitated by allocation. As can be seen from Fig. 2, part 1 is processed on both of the machines. Since there needs to be a transfer from machine 2 to machine 1, which corresponds to a movement from station 6 to station 5; all of the nodes 1_3, 1_6, 1_5 and 1_2 are used in the tour of Fig. 5.

TSP is a well known NP-Hard problem. The formulation above differs from a classical TSP formulation in that some nodes could be unvisited and costs depend on waiting times which are variables. As such, it is more general than the classical TSP formulation and requires a great amount of computational effort even if the number of machines in the cell is small. This is expected since the problem considered in this paper generalizes the classical problem of parallel machine scheduling with the objective of minimizing makespan which has been shown to be NP-complete even in the simplest 2-machine case (see for example Hall et al., 2000; Lenstra et al., 1977). Consequently, we focused our attention on heuristic approaches and composed the algorithms introduced in Section 4.

4. Heuristic solution methodology

We propose a two-stage algorithm in which the second stage works as an improvement phase on the first one. Throughout the solution procedure we consider *machines* instead of *stations* and use m instead of k, where $m=1$ corresponds to $k=2,5$ and $m=2$ corresponds to $k=3,6$.

In the first stage, we find a solution without any allocation, as is the case in a parallel machine system. The aim is to minimize blocked times. In contrast to a waiting time, a blocked time occurs when processing is completed before the robot arrives at the machine. It is represented as follows:

$$b_{j_m}=\max\{0,v_{j_m}-P_j\}, \quad j=1,\ldots,n, \ m=1,2, \tag{17}$$

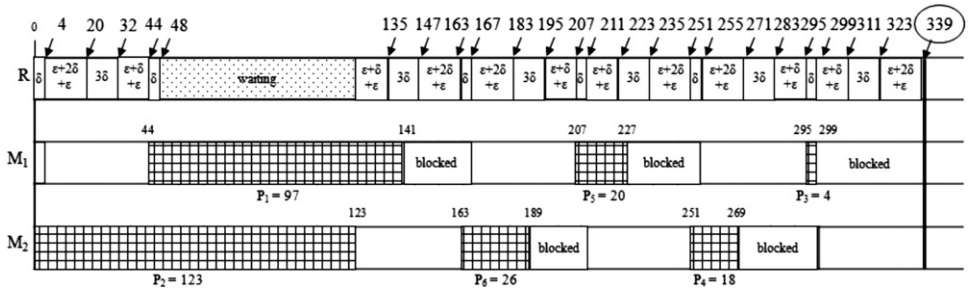

Fig. 6. Gantt-chart of LPT approach for Example 2.

where j is the part loaded on machine m and v_{j_m} is the total activity time of the robot in between just after loading machine m by part j and arriving in front of the machine to unload it.

Considering Eqs. (1) and (17) together, it can be seen that there is a strong relation between waiting and blocked times. In particular, $w_{j_m} \cdot b_{j_m} = 0$, where j is the part loaded on machine m.

In the second stage of our algorithm, we search for improvements by considering allocation alternatives. With the methodology to be prescribed in Section 4.2, we choose a part that can be processed on both of the machines. It will be apparent later that it is enough to only consider one such part. At the end of this stage, we have a new solution for which we know the parts to be processed on each of the machines, their sequences, and related processing time values.

In parallel machine systems, LPT approach is commonly used (Blazewicz et al., 2001). This rule sorts the jobs in the order of decreasing processing times. Whenever a machine becomes free, the job with the largest processing time which is ready at the time starts processing. This rule schedules the longest jobs first so that no one large job will 'stick out' at the end of the schedule and dramatically lengthen the completion time of the last job. The time complexity of LPT is known to be $O(n \log n)$ (Blazewicz et al., 2001). In order to describe the LPT approach, we use the following example with no allocation.

Example 2. Assume that we have six parts to be completed with corresponding processing times: $P_1 = 97$, $P_2 = 123$, $P_3 = 4$, $P_4 = 18$, $P_5 = 20$, and $P_6 = 26$. ϵ and δ are both four units of time. The related Gantt-chart is given by Fig. 6.

In this example, the system starts by processing the longest part and continues with the next longest one at each step until no more parts are left. It is possible to use either 1-MPS or more MPS's in order to reach to a steady state which defines a cyclic solution. LPT helps to provide equal work-loads for machines. Since our problem environment is also a special case of parallel machine systems, we will compare our results with the ones obtained by using the LPT rule.

4.1. Stage 1: construction algorithm

In Stage 1, we initially ignore the possibility of allocation for all the parts to be produced. The main aim is to minimize blocked times, by which we will be able to obtain a good assignment as well as a sequence. The importance of blocked time values arises from the importance of equal work-loads on machines; when we minimize blocked times on machines, it is more likely to obtain a fair assignment. We also checked the solutions obtained when waiting time values are considered as the first priority and noticed that better solutions are obtained for almost all examples when minimizing the blocked times is the first priority. We consider the waiting times only as tie-breakers. One of the main assumptions of this stage is that we consider only non-delay schedules. In a non-delay schedule, the machine to load is always the one that is just unloaded. The exception is during the placement of the first two parts since we start to solve the problem when the system is empty.

We use the following additional notation throughout the proposed algorithms:

t_{now}	current time of the system following the activities of the robot.
S	set of parts available in the input buffer.
S_1, S_2	sets of parts assigned to machines 1 and 2 respectively.
m	selected machine, where $m=0,1,2,3$.
pos	current position of the robot, where pos$=0,1,2,3$.
s_{j_m}	starting time of the processing of part j on machine m, where $j=1,\ldots,n$ and $m=1,2$.
F_m	completion time of the currently loaded part on machine m, where $m=1,2$.

These definitions are strongly related to the ones given in Section 3. We use t_{now} following the robot activities which were defined by $comp_{i_k}$ values previously, and keeping the result cumulatively. s_{j_m} corresponds to $start_{j_k}$ values where m corresponding to machines is used instead of k corresponding to stations. F_m is simply the sum of the start time and the processing time of the currently loaded part on machine m.

Pseudo-codes of the two algorithms, where the first one describes Stage 1 applied for the first set of parts and the second one, the part selection procedure, are given by Algorithms 1 and 2.

Algorithm 1. First MPS solution.

Input: $S=\{1,\ldots,n\}$, ϵ, δ, $P_1,\ldots,P_n$.
Output: S_1, S_2, w_{j_1}, w_{j_2}, b_{j_1}, b_{j_2}.
$S_1=\emptyset$, $S_2=\emptyset$,
$w_{j_1}=0$, $w_{j_2}=0$,
$b_{j_1}=0$, $b_{j_2}=0$,
$t_{now}=0$, pos$=0$, $F_1=0$, $F_2=0$,
while $S\neq\emptyset$ **do**
  if $S_1=\emptyset$
    $m=1$,
    $S_1=S_1\bigcup\{1\}$, $S=S-\{1\}$,
    $t_{now}=t_{now}+2\epsilon+\delta$,
    pos$=1$,
    $s_{1_1}=t_{now}$, $F_1=s_{1_1}+P_1$,
    $current_1=1$,
  end if
  if $S_2=\emptyset$ **then**
    $m=2$,
    $t_{now}=t_{now}+$posδ,
  else
    if $F_1\leq F_2$ **then**

21: $m=1$,
22: **else**
23: $m=2$,
24: **end if**
25: $t_{now}=t_{now}+|\text{pos}-m|\delta$,
26: $w_{current_{m\,m}}=\max\{0,F_m-t_{now}\}$,
27: $t_{now}=t_{now}+w_{current_{m\,m}}+\epsilon+(3-m)\delta+\epsilon+3\delta$,
28: **end if**
29: **PartSelect**,
30: $best \leftarrow$ Part returned by **PartSelect**,
31: $t_{now}=t_{now}+2\epsilon+m\delta$,
32: $S_m=S_m\bigcup\{best\}$, $S=S-\{best\}$,
33: $\text{pos}=m$,
34: $s_{best_m}=t_{now}$, $F_m=s_{best_m}+P_{best}$,
35: $current_m=best$
36: **end while**

As it is given in line (3) of Algorithm 1, initially both machines are empty and all the parts in an MPS are ready to be processed. At this initial state, the robot is in front of the input buffer; line (6). Since the first machine is empty as in line (8), the first part (according to the order in the input buffer) is taken from the input buffer and loaded on this machine. This movement is not restrictive since the solution obtained when we solve the problem assigning the first part on the second machine instead of the first is simply the mirror image of the one that is attained. The next machine selected is going to be the second one, since it is empty in line (16).

At this point, which refers to lines (29) and (30) of Algorithm 1, the part to be loaded is determined by Algorithm 2 according to the possible blocked and waiting times caused by the remaining parts. The word 'possible' indicates that at any decision point, blocked and waiting time values of each part left in the input buffer are calculated as if it is the one that is going to be loaded next, for which related steps are described by Algorithm 2 as is mentioned in lines (6)–(18). Comparing these values we first check blocked times and pick the one causing smallest blocked time value, lines (19)–(22). In the event of equivalence of blocked times, we compare waiting times in lines (23)–(29). If waiting times are also the same, we pick any one of the parts randomly. Throughout the algorithm, $\overline{B}$ and $\overline{W}$ are the respective blocked and waiting times corresponding to the best part choice (j) at hand. Initially, they are set to very large numbers.

Returning back to Algorithm 1, after loading the selected part on machine 2, we check if all the parts are assigned or not. If not, this means that we are still in the while loop and go back to line (7) for the assignments of the remaining part(s). Since both machines are not empty we move to line (19) and from now on the decision of the machine to unload is taken according to the times that the loaded parts are to be completed as in lines (20)–(24). After unloading the selected machine, we determine the part to load on it according to Algorithm 2 again. Machine and part selections and load/unload steps are repeated as long as there are parts in the input buffer. When all the parts in set S are assigned and sequenced by Algorithm 1, the algorithm either calculates the cycle time value repeating the assignments and sequences obtained so far, or loads a new set and goes on the part selection steps for this new set.

Algorithm 2. PartSelect.

1: $\overline{B}=M$, $\overline{W}=M$,
2: **for** $i\in S$ **do**
3: $t_{now}=t_{now}+2\epsilon+m\delta$,
4: $\text{pos}=m$,
5: $s_{i_m}=t_{now}$, $F_m=s_{i_m}+P_i$,
6: **if** $F_m\le F_{(3-m)}$ **then**
7: $w_{i_m}=\max\{0,P_i-t_{now}\}$,
8: $t_{now}=t_{now}+w_{i_m}+4\epsilon+6\delta+\delta$,
9: $b_{i_{|1-m|}}=\max\{0,t_{now}-F_{(3-m)}\}$
10: $B=b_{i_m}$, $W=w_{i_m}$,
11: **else**
12: $t_{now}=t_{now}+\delta$,
13: $\text{pos}=(3-m)$,
14: $b_{i_{|1-m|}}=\max\{0,t_{now}-F_{(3-m)}\}$,
15: $t_{now}=t_{now}+\max\{0,F_{(3-m)}-t_{now}\}+4\epsilon+6\delta+\delta$,
16: $b_{i_m}=\max\{0,t_{now}-F_m\}$
17: $B=b_{i_{|1-m|}}+b_{i_m}$, $W=w_{i_m}$,
18: **end if**
19: **if** $B<\overline{B}$ **then**
20: $\overline{B}=B$,
21: $\overline{W}=W$,
22: $j=i$,
23: **else if** $B=\overline{B}$ **then**
24: **if** $(W<\overline{W})$ **then**
25: $\overline{B}=B$,
26: $\overline{W}=W$,
27: $j=i$,
28: **end if**
29: **end if**
30: **end for**
31: **return** j

Since we try to reach to a steady-state cyclic solution, end-up conditions for the algorithm are either to reach to a state that the first part is assigned to the same machine while other machine's state is also the same as the previous or to repeat the algorithm for a given number of MPS's. 'Given number of MPS's' means that we load input buffer with the same MPS for a few times and run the algorithm trying to reach to a state that was observed before. If we can find such a point at the end of Algorithm 1, algorithm stops without repeating the given number of MPS's; but if we could not find such a steady state solution even after a given number of MPS sets, algorithm does not search any more and takes the resulting state as the starting one. As can be guessed, the state that we look for reaching for the second time is the load of the first part on the same machine; since load of it is the starting point of each set.

End-up policy can be understood better using Fig. 7. In chart 'a', the system returns to its initial state after loading only one MPS, whereas in chart 'b' the initial state is reached after two loads of the same MPS. In chart 'b', if we take 1 as the maximum MPS number, i.e. we assume that we cannot load the set more than once, our system would stop at point '1' and we would obtain the solution given by chart 'c' instead of the one in 'b'. In consequence of this policy, we obtain solutions producing either 1 or higher number of MPS's (mostly $2-MPS$). However, in our solutions, a $2-MPS$ solution does not refer to a solution of two times the part number in $1-MPS$; we use this term when we load a second MPS after the first one is completed while these two sets do not necessarily use the same sequences. This means that our choice of $2-MPS$ does not cause any need for extra inventory. A step-by-step detailed explanation of the construction algorithm is given on a numerical example in Appendix A. We will now give the complexity of this method.

Lemma 1. *The time complexity of the Construction Algorithm is* $O(n^2)$.

Proof. Throughout the algorithm, we assign the parts on machines as defined above. Starting from the second part, we

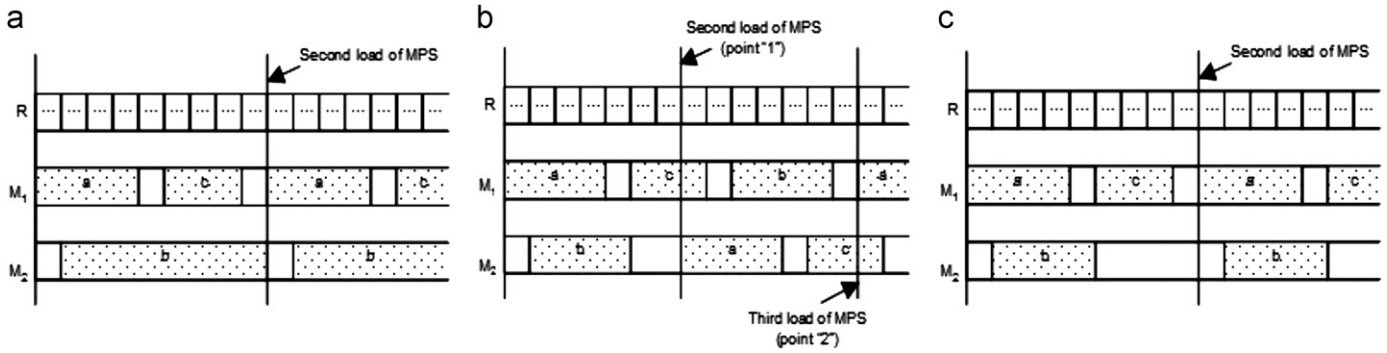

Fig. 7. End-up policy.

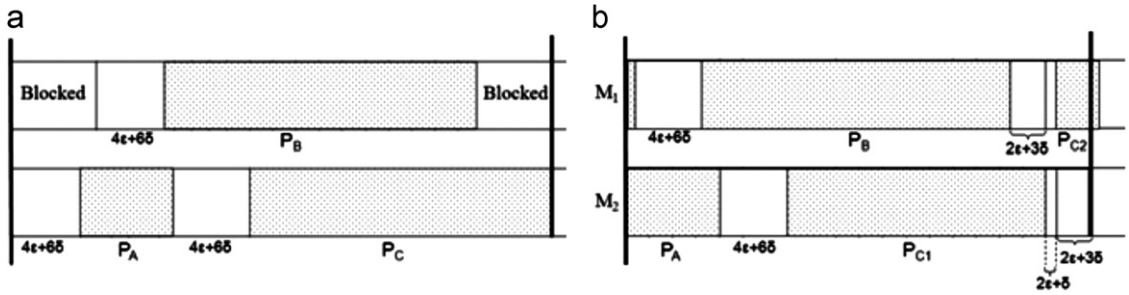

Fig. 8. Part selection policy for Stage 2.

check all the remaining parts for this decision. In other words; we consider $n-1$ possibilities for the second part, $n-2$ possibilities for the third one, …and 2 possibilities for the $(n-1)$ th part. Since the sum of these values $(n-1,n-2,\ldots 2)$ equals to $n^2/2-n/2-1$, complexity equals to $O(n^2)$. □

4.2. Stage 2: allocation algorithm

Allocation is a primary issue with this study. Although, for the first stage, we assume the parts to be processed on only one of the machines; for this second stage, we try to find better solutions by letting one part to be processed on both machines. In order to reach our goal, we first decide on the part to allocate and then determine the processing time values on each machine for this allocated part. For the first decision, looking at the total blocked time values of machines (B_m) from the solution obtained in Stage 1, we select the machine that has larger blocked time as the one that extra processing time will be added (machine x). The part to allocate (part i) will be the one that causes the largest blocked time value on machine x among the parts processed on the other machine (machine y in the initial solution, where $y \neq x$). Letting $b^*_{j_m}$ be the blocked time values observed on machine m in the same time interval with the processing and unload of part j, we pick part a to allocate where $B_a = \max\{b^*_{j_m}\}$, $m \neq x$. This part selection policy is depicted in Fig. 8.

In the first chart of the figure, part C is the one causing the largest blocked time (which is also the only blocked time) on the first machine, thus it is chosen to be allocated. After loading the part on the second machine and unloading the first machine as before, we unload part C without waiting for all the processing to complete and load it on the first machine to perform the rest of its processing. As can be seen, the allocation shortened the cycle time value while filling the blocked time observed in the initial solution.

After choosing the part, we should determine how we allocate the processing times on each machine. We know that the total processing time of part i is P_i and need to calculate P_x which determines the processing time to be allocated on machine x.

$4\epsilon+4\delta$ is the extra time for machine x since the machine was not supposed to work for that part in the solution obtained by Stage 1. When we allocate a part, following movements will be performed immediately after the unload of machine x: go to machine y from the output buffer (δ or 2δ, for $x=1$ or $x=2$ respectively), unload the part to be allocated from machine y (ϵ), go to machine x (δ), load the part on machine x (ϵ), unload the part (ϵ), go to the output buffer (2δ or δ, for $x=1$ or $x=2$ respectively), and load the part on output buffer(ϵ).

$2\epsilon+4\delta$ is a twofold important value. First, it represents the time that passes between the load of the allocated part on machine y and the time that robot comes back to unload it after unloading the previous assigned part on machine x which means to get machine x ready for the allocated part [travel time to machine x from machine y (δ), unload of part from machine x (ϵ), travel time to output buffer (2δ if machine x is the first one or δ if machine x is the second one), load of part on output buffer (ϵ), and travel time to machine y (2δ if machine y is the first one or δ if machine y is the second one)]. Secondly, it corresponds to the time that passes between the load of the allocated part on machine x and the time that robot comes back to unload it after loading the next assigned part on machine y [travel time to input buffer from machine x (δ if machine x is the first one or 2δ if machine x is the second one), unload of part from input buffer (ϵ), travel time to machine y (δ if machine y is the first one or 2δ if machine y is the second one), load of part on machine y (ϵ), and travel time back to machine x (δ)].

The following result is imperative in the choice of allocation.

Lemma 2. *Allocation is useful if and only if there exists a P_x value satisfying the following conditions*:

(*a*) $P_x \leq B_d-(4\epsilon+4\delta)$,

(*b*) $\max\{0,P_i-P_x-2\epsilon-4\delta\}+\max\{0,2\epsilon+4\delta-P_x\} \leq B_a-P_x$,

(c) $0 \le P_x \le P_i$. (18)

Where P_i is total processing time of the chosen part, P_x is the time we need to determine, B_d is the difference of total blocked times of the machines, and B_a is the blocked time on machine x that is caused by the chosen part.

Proof. For the first relation, the significance of $4\epsilon+4\delta$ arises from the fact that it is a newly added time for machine x which can be seen as the fixed cost of allocation for machine x. The sum of this value and the allocated processing time on machine x (P_x) should not be more than the blocked time difference (i.e. the excess of blocked time on machine x). Otherwise, the new cycle time will become worse. This value can be traced from Fig. 8, as the sum of $2\epsilon+3\delta$ and $2\epsilon+\delta$. We observe from the same figure that this condition needs to be considered when determining P_x values as the cycle time can be shortened only if the sum $P_x+4\epsilon+4\delta$ is less than the blocked time value.

For the second relation, it was mentioned before that $2\epsilon+4\delta$ is the time needed for unload of machine x and travel to machine y for the allocated part. If the process left on machine y is not completed by the time the robot arrives at the machine, in other words, if processing time value is larger than $2\epsilon+4\delta$; waiting time will occur for machine y which is equivalent to blocked time for machine x. In a similar manner, $2\epsilon+4\delta$ also represents the time needed for taking the allocated part from machine y to machine x and coming back to this machine after loading a new part on machine y. If the process assigned to machine x is already completed by the time the robot arrives at the machine, in other words, if processing time value is smaller than $2\epsilon+4\delta$; blocked time will occur for machine x. In particular, the left hand side of the inequality gives us the total blocked time value observed on machine x for the allocated part after allocation is performed. We know that B_a is the blocked time caused by the chosen (to be allocated) part and P_x is the allocated time value. So, the right hand side of the inequality gives us the available time that can be used for other movements after the part is assigned. These observations can be seen in Fig. 9. In essence, the blocked times occurring after allocation is performed should not be more than the initial solution. □

By the conditions defined in Lemma 2, we obtain a range for the P_x value. We consider two values among the ones satisfying the above conditions to select the P_x value; (i) we could take the one in the middle of the range obtained, or (ii) we could take the one closest to $2\epsilon+4\delta$, in order to obtain minimum blocked time value for machine x. In most cases, both of these two choices gave the same result; however, since we obtained better results by the second one for the remaining cases, we use the value closest to $2\epsilon+4\delta$.

If there does not exist any part that satisfies the conditions of Lemma 1, there is no need to perform an allocation. Else, we follow the same assignments obtained in Stage 1 until part j is loaded on machine y. Once part j is loaded (with a processing time value of (P_j-P_x) instead of P_j); we go to machine x, get prepared for part j by unloading the machine and without loading the next assigned part, we go to machine y, wait if necessary, unload this machine and take the allocated part to machine x (with a processing time value of P_x). After this point, we go back to the assignments of Stage 1 again; go to the input buffer, take the next assigned part for machine y, go to the machine and load the part. Using the same machine selection policy described for Stage 1, and following the assignments obtained in Stage 1; we perform load/unload activities for all the parts. In order to find a steady state solution, the sequences determined so far are repeated until two successive results are found to be the same.

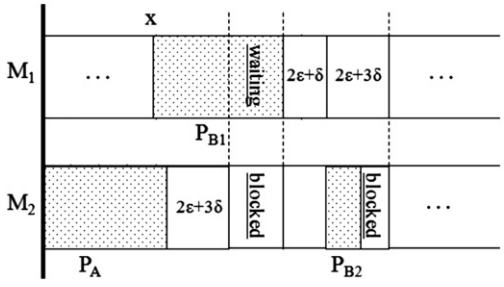

Fig. 9. Effect of the value $2\epsilon+3\delta$ on allocation.

Lemma 3. *The time complexity of the allocation algorithm is $O(n)$.*

Proof. We determine the part to allocate according to the conditions in Lemma 2. Since the necessary calculations are performed once for each of the n parts, they take a total of $O(n)$ time. □

One of the main points for this stage is that; we are not guaranteed that we will be able to find a better solution with allocation. Besides, the success of this method depends mostly on the choice of assignments and sequence of the previous stage. Similar to Stage 1, a detailed explanation of the allocation algorithm is given on a numerical example in Appendix B.

5. Computational experiments

In this section, the algorithms described so far are compared with the LPT approach. In order for our results to be conclusive, we have run the algorithm on a wide range of instances. Performing the solution procedures over preliminary experiments, we first observed that five factors, which are the five basic parameters of this study, play important roles on the solutions obtained. In order to set appropriate values for each factor, we run further extensive trials for each of the criteria. As a result of these studies, we obtained the factors and factor levels shown in Table 2.

Throughout this study, we use the following equation in order to calculate the improvements:

$$\%\ \text{improvement over LPT} = \frac{\text{C(LPT)}-\text{C(Stage1)}}{\text{C(LPT)}} \cdot 100, \quad (19)$$

where 'C(LPT)' is the cycle time obtained by the LPT approach and 'C(Stage1)' is the one obtained by the Stage1 approach.

We have taken 10 replications for each factor combination, resulting in 1620 randomly generated runs. All of these instances are generated uniformly in the respective intervals. When we compare the Stage1 solutions (no allocation) with the LPT approach, we see that only 3 out of 1620 give better results with LPT. Average gain of our algorithm is 2.67%, where this value is up to 28.3% for a case with high part number, high mean, medium range, low epsilon, and low delta levels. Improvements for the factor levels are given in Table 3.

Table 2
Factor and factor levels.

Factor	Number of levels	Low	Medium	High
Number of parts	3	≤ 10	$11 \le \cdots \le 99$	$100 \le \cdots \le 800$
Average processing times	2	≤ 50	–	$51 \le \cdots \le 4800$
Range of processing times	3	≤ 139	$140 \le \cdots \le 149$	$150 \le \cdots \le 640$
Epsilon	3	1,2,3	4,5,6	7,8,9
Delta	3	1,2,3	4,5,6	7,8,9

In the problem that is considered throughout this study, we have three basic time elements: processing times, load/unload times, and travel times. Different from the LPT approach, our construction algorithm takes the robot movements, i.e. load/unload and especially travel times, into account. As a result of this fact, our algorithm gives better solutions for almost all cases considered. Looking at the factors and factor levels in a more detailed manner, we can make more specific comments over the effectiveness of our algorithm. As can be seen from Table 3, range is one of the critical factors. It has turned out that for a high range factor setting our gain is 4.49% on the average. We attribute this to the increase in the number of alternative solutions. When the range of processing times is low, LPT is able to work well; however when the range gets higher, LPT gets weak and our construction algorithm performs better as it takes the relationship between the processing times and the robot movements into consideration. For those with small number of parts, we also get better solutions than the LPT approach, since the solution quality is more vulnerable to the myopic decisions.

During comparisons with Stage 2 solutions, we used the same factors and factor levels. One of the basic points here is that, not every solution could be improved by allocation. As a result of this fact, out of 1620 runs, only 235 of LPT solutions and 35 of heuristic solutions can be improved. Average improvements observed for these two approaches are 1.23% and 1.62%, respectively, as summarized in Table 4. Although improvement percentages are higher when Stage 2 is applied over Stage 1, the number of improved solutions are significantly higher for LPT results. This situation arises from the effectiveness of Stage 1 results; as we have obtained already better solutions applying Stage 1 rather than the LPT approach, improvement chance is lower in this case.

In a similar manner to the previous observations, we search for the most effective factors and factor levels for the problem. As can be seen from Table 4, the gain of the algorithm is up to 9.90% over an LPT solution for a case with low part number, high mean, high range, low epsilon and high delta levels. Low number of parts is able to react better to allocation for both LPT and heuristic approaches. As the part number increases, the positive effect of allocation fades away since the allocation makes the already high number of robot movements even higher. The range of processing times also plays an important role in Stage 2, similar to Stage 1. As is mentioned before, LPT solutions are worse especially for high ranges. Observing mean and range factors together, we found out that for high average processing time and high range levels, allocation is a preferable option on 72 out of 270 instances and the average gain for these trials is 1.7%. Considering allocation possibility as an improvement algorithm, we are able to increase the quality of the weaker solutions. Although there is a slight difference between the three levels, lower epsilon values are more preferable for the allocation decisions since the disadvantage of loading/unloading the allocated part twice gets smaller as the epsilon values are small. Furthermore, as expected due to the conditions stated in Lemma 2, allocation decisions are affected by the delta values. When activities of the robot are ignored in a schedule, it becomes bottleneck leading to higher percentages for higher delta values. Performing allocation, the robot movements are spread out over the total cycle time in a more effective way and better solutions are easily obtained.

For the same problem, we checked the possibility of allocating two parts instead of one. We tried to allocate a second part after

Table 3
Improvement for Stage 1.

Factor	Factor level	Average (%)	Minimum (%)	Maximum (%)
Part #	Low	4.32	0.60	24.30
	Medium	2.16	0.00	14.50
	High	1..54	0.00	28.30
Mean	Low	2.12	0.00	19.40
	High	3.23	0.00	28.30
Range	Low	0.80	0.00	10.30
	Medium	2.72	0.00	28.30
	High	4.49	2.00	24.30
Epsilon	Low	2.62	0.10	28.30
	Medium	2.78	0.00	24.30
	High	2.62	0.00	18.20
Delta	Low	2.82	0.20	28.30
	Medium	3.05	0.20	17.90
	High	2.14	0.00	19.40

Table 4
Observations for Stage 2.

Factor	Factor level	Over LPT				Over Stage1			
		Number[a]	Improvement[b]			Number[a]	Improvement[b]		
			Average (%)	Minimum (%)	Maximum (%)		Average (%)	Minimum (%)	Maximum (%)
Part #	Low	42	3.39	0.30	9.90	4	6.65	6.00	7.20
	Medium	131	1.03	0.00	5.10	22	1.21	0.20	2.20
	High	62	0.20	0.00	0.50	9	0.37	0.20	0.50
Mean	Low	76	1.58	0.00	7.20	16	2.02	0.20	7.20
	High	159	1.07	0.00	9.90	19	1.28	0.20	6.00
Range	Low	20	0.82	0.00	3.70	6	2.40	1.40	6.00
	Medium	87	0.50	0.00	2.50	24	1.67	0.20	7.20
	High	128	1.80	0.10	9.90	5	0.44	0.20	0.70
Epsilon	Low	78	1.30	0.00	9.90	5	2.50	0.50	7.20
	Medium	77	1.15	0.00	8.30	11	1.41	0.20	7.10
	High	80	1.25	0.00	8.40	19	1.51	0.30	6.30
Delta	Low	50	0.88	0.00	4.30	1	0.30	0.30	0.30
	Medium	74	1.26	0.00	8.40	3	3.03	0.30	6.30
	High	111	1.38	0.00	9.90	31	1.52	0.20	7.20

[a] Number of improved solutions out of 1620 / (number of levels of the factor) instances.
[b] Average/minimum/maximum improvement percentages for the given number of improved solutions.

the first one is allocated but could not find any P_x value satisfying the conditions defined by Lemma 3 for any of the 1620 solutions. This was also an expected result, since we define the constraints trying to take the advantage of blocked times and to balance the work-loads on the machines as much as possible.

We also checked the results obtained by choosing the part causing the second largest blocked time value on machine x among the ones processed on machine y, to allocate. For the LPT solutions, 85 out of 1620 changed; 41 of them gave the same result with different parts and processing time values and 44 of them could not perform allocation while they did before. Similarly, for the initial algorithm's solutions, 38 out of 1620 changed; 23 of them gave the same result with different parts and processing time values, 14 of them could not perform allocation while they did before and only 1 of them gave a smaller solution with a percentage of 18%. This was another expected result in light of the second condition given in Lemma 1. By selecting a part with a smaller blocked time value, we decrease the right hand side of this inequality, while other conditions remain the same, and obtain a smaller range for P_x value. Therefore, it becomes less likely to find such a solution.

6. Conclusion

In this study, our primary focus is on an *NP*-hard scheduling problem arising in two-machine manufacturing cells which repeatedly produce a set of multiple part-types, and where transportation of the parts between the machines is performed by a robot. Since the parts considered are not identical, we need to solve both the problems of sequencing the parts and the robot moves for robotic cells. Taking the advantage of the flexibility property, processing times of parts on each machine are not assumed to be constant as is typically done in the existing literature. Due to this, allocation becomes our third problem. We first modeled this problem as a traveling salesman problem (TSP) in which the distance matrix consists of decision variables as well as parameters. The formulation obtained is more general than the classical TSP formulation and requires a great amount of computational effort even if the number of machines in the cell is small. Consequently, we focused our attention on heuristic approaches. We first constructed an algorithm which does not allow allocation for parts. We compared the results obtained with this algorithm with the well-known LPT approach, and noticed that our algorithm outperformed for almost all the examples in our experimental design. We then considered the case of allocation and let the robot load a selected part on both of the machines consecutively. Although there would be extra load/unload and travel times, decreasing the waiting times and using the machine capacities more effectively, allocation gave better results for some specific cases. As far as the authors know, this is the first study to consider allocation possibility in multiple part-type robotic cell scheduling literature.

Appendix A. Numerical example for construction algorithm

We will use Example 2 to describe the construction algorithm. The algorithm starts by loading the first part on the first machine; thus

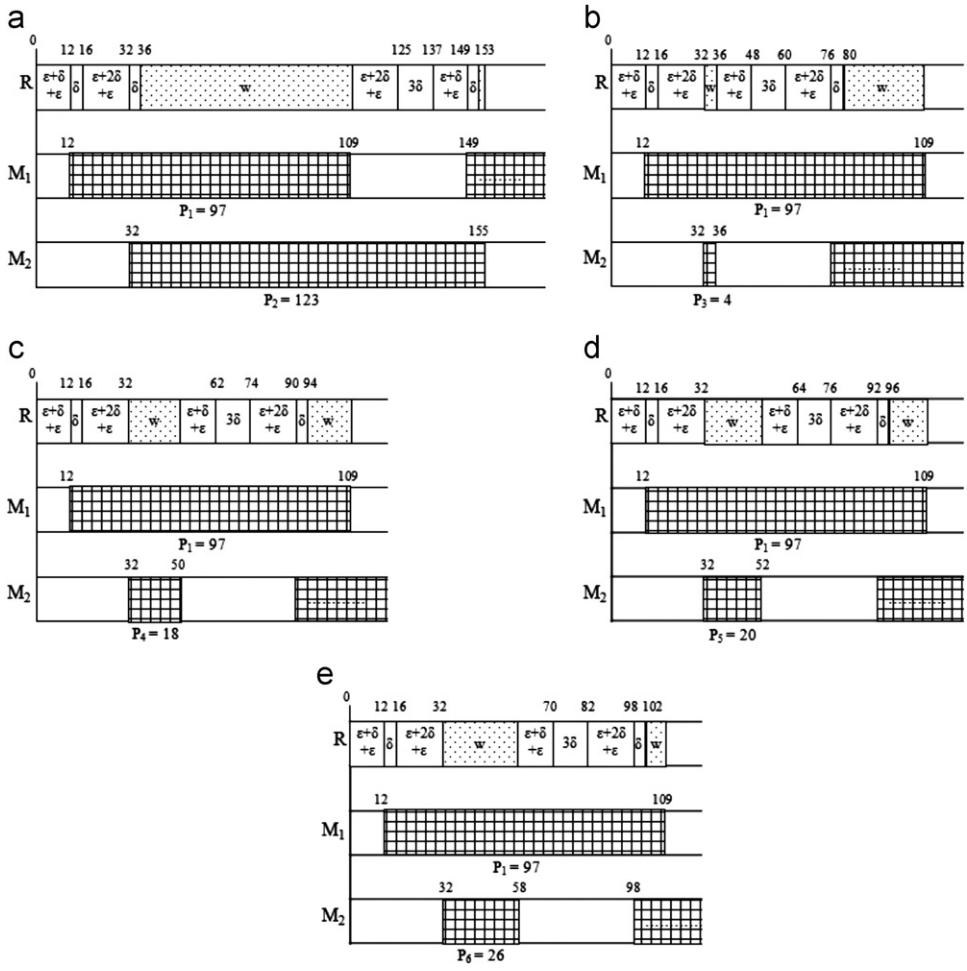

Fig. 10. Part selection for Example 1 in Stage 1.

robot loads P_1 on the first machine, which will take a total of $2\epsilon+\delta=2(4)+4=12$ units of time. Time of the system is 12 and robot position is pos = 1. Since the second machine is empty, it is chosen to be the machine to load and robot goes to the input buffer (posδ). Time of the system is $t_{now}+\text{pos}\delta=12+1(4)=16$ and robot position is pos = 0. Start and finish times of the part are as follows: $s_{i_m}=s_{1_1}=t_{now}=12$, and $F_m=F_1=s_{i_m}+P_i=s_{1_1}+P_1=12+97=109$.

Now we need to determine which part to load, thus we will calculate possible blocked and waiting time values for the remaining parts (i.e. parts 2 and 3). These observations can also be followed from Fig. 10.

(a) If we select part 2 ($P_2=123$), following situations will be observed:

Robot will take the part from input buffer (ϵ), go to the second machine ($|0-2|=2\delta$), load the machine (ϵ). Robot position is pos = 2 and system time will be $t_{now}+2\epsilon+m\delta=t_{now}+2\epsilon+2\delta=16+2(4)+2(4)=32$. Start and finish times of the part are as follows: $s_{i_m}=s_{2_2}=t_{now}=32$, and $F_m=F_2=s_{i_m}+P_i=s_{2_2}+P_2=32+123=155$.

Since both machines are loaded and $F_1\le F_2$, unload will be performed for the first machine. Robot will go to the machine ($|2-1|\delta=\delta$), wait until processing of the part is completed (i.e. until $F_1=109$, meaning 73 time units of wait), take the part (ϵ), go to the output buffer ($|1-3|\delta=2\delta$), and leave the part (ϵ); these will make a total of $w_{1_1}+2\epsilon+3\delta=73+2(4)+3(4)=93$ and system time will be $32+93=125$. Robot position will be pos = 3. Robot will then go to the input buffer ($|3-0|\delta=3\delta$), take a new part (which has not been determined yet) (ϵ), go to the first machine ($|0-1|\delta=\delta$), load the part (ϵ). System time will increase by $2\epsilon+4\delta=2(4)+4(4)=24$ and will be $125+24=149$.

Although we do not know the processing time of the newly loaded part, we check at this point if there is any waiting or blocked time for machine 2. Robot goes to the second machine which will make system time $149+|1-2|\delta=149+1(4)=153$ and this time is compared with the finish time of the already loaded part. F_2 is known to be 155, so we have $\max\{0,F_2-t_{now}\}=\max\{0,155-153\}=2$ units of waiting time at this moment.

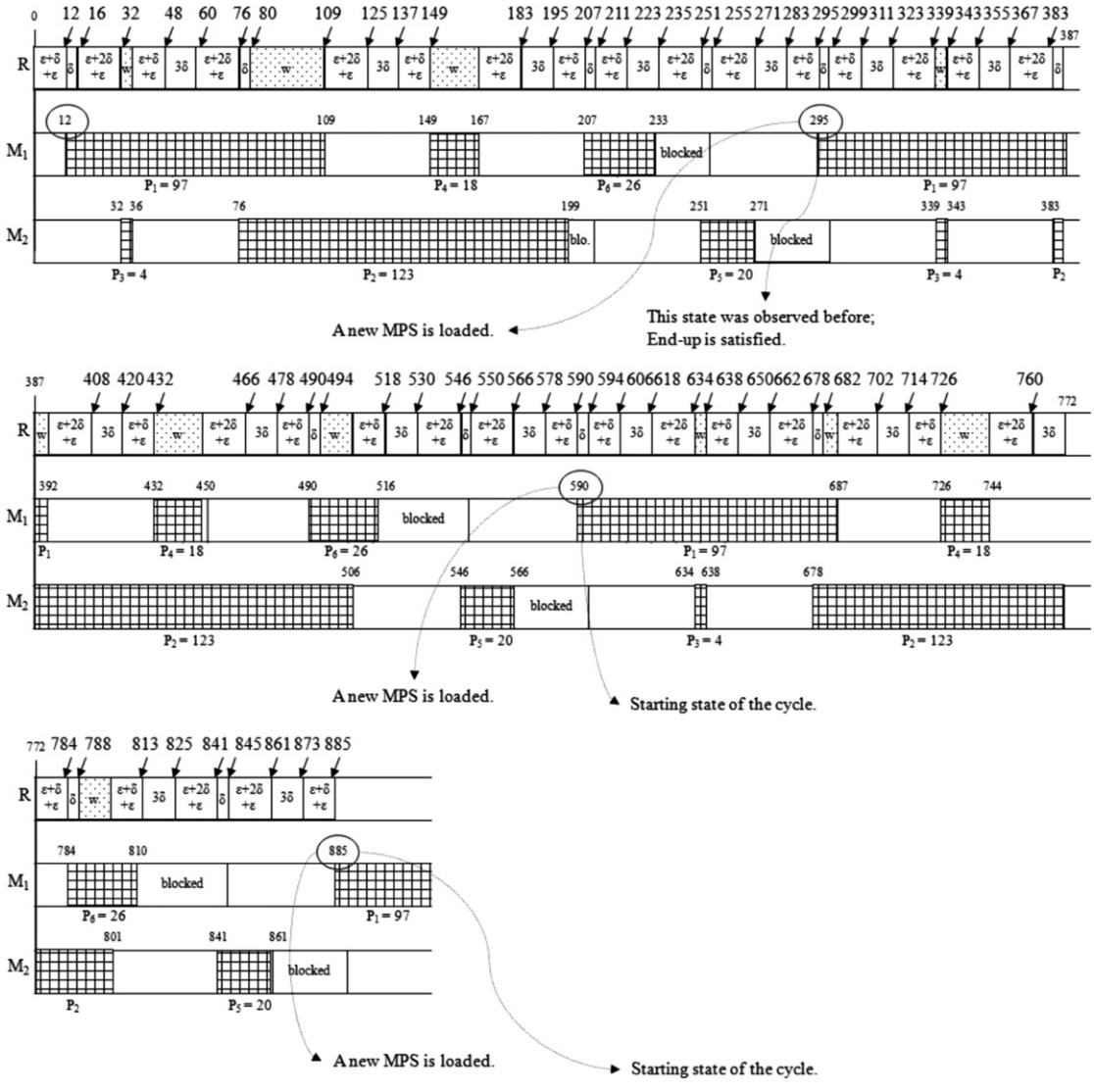

Fig. 11. Gantt-chart of Stage 1 for Example 2.

As a result, this choice will cause 75 units of waiting and 0 units of blocked time.

With the help of similar observations for all the remaining parts, we obtain the following results:

(b) If we select part 3 ($P_3 = 4$), this choice will cause $4+29 = 33$ units of waiting and 0 units of blocked times.
(c) If we select part 4 ($P_4 = 18$), this choice will cause $18+15 = 33$ units of waiting and 0 units of blocked times.
(d) If we select part 5 ($P_5 = 20$), this choice will cause $20+13 = 33$ units of waiting and 0 units of blocked times.
(e) If we select part 6 ($P_6 = 26$), this choice will cause $26+7 = 33$ units of waiting and 0 units of blocked times.

Since all the possible blocked and waiting time values are determined to be the same, robot will select part 3 arbitrarily and take the part from input buffer, go to the second machine, load the machine, wait until processing of the part is completed, take the part, go to the output buffer, leave the part and go to the input buffer for the new part. This point is again a decision point for the algorithm. According to the possible blocked and waiting time values again, this time robot picks part 2 and loads on the second machine. These part selection steps continue until all the parts are assigned. By the time we have finished the assignments of all the parts in the first MPS, we have also reached to a state that was observed before; so we end-up the algorithm having obtained the sequence and the schedule. Now we will determine the cycle time value of this result by repeating the assignments and the order until we reach to a steady state solution. All the movements and processes of this example can be seen from Fig. 11.

The time that passed until the end-up point is $295-12 = 283$. We repeat the determined sequence and scheduled movements and reach the same state at time 590, for which $590-295 = 295$ units of time have passed. Third repetition finished at time 885, by which we have counted again $885-590 = 295$ units of time. Thus, the solution is reported as 295.

Appendix B. Numerical example for allocation algorithm

We again consider Example 2, for which Gantt-charts of LPT and Stage1 solutions are given in Section 4 by Figs. 6 and 11,

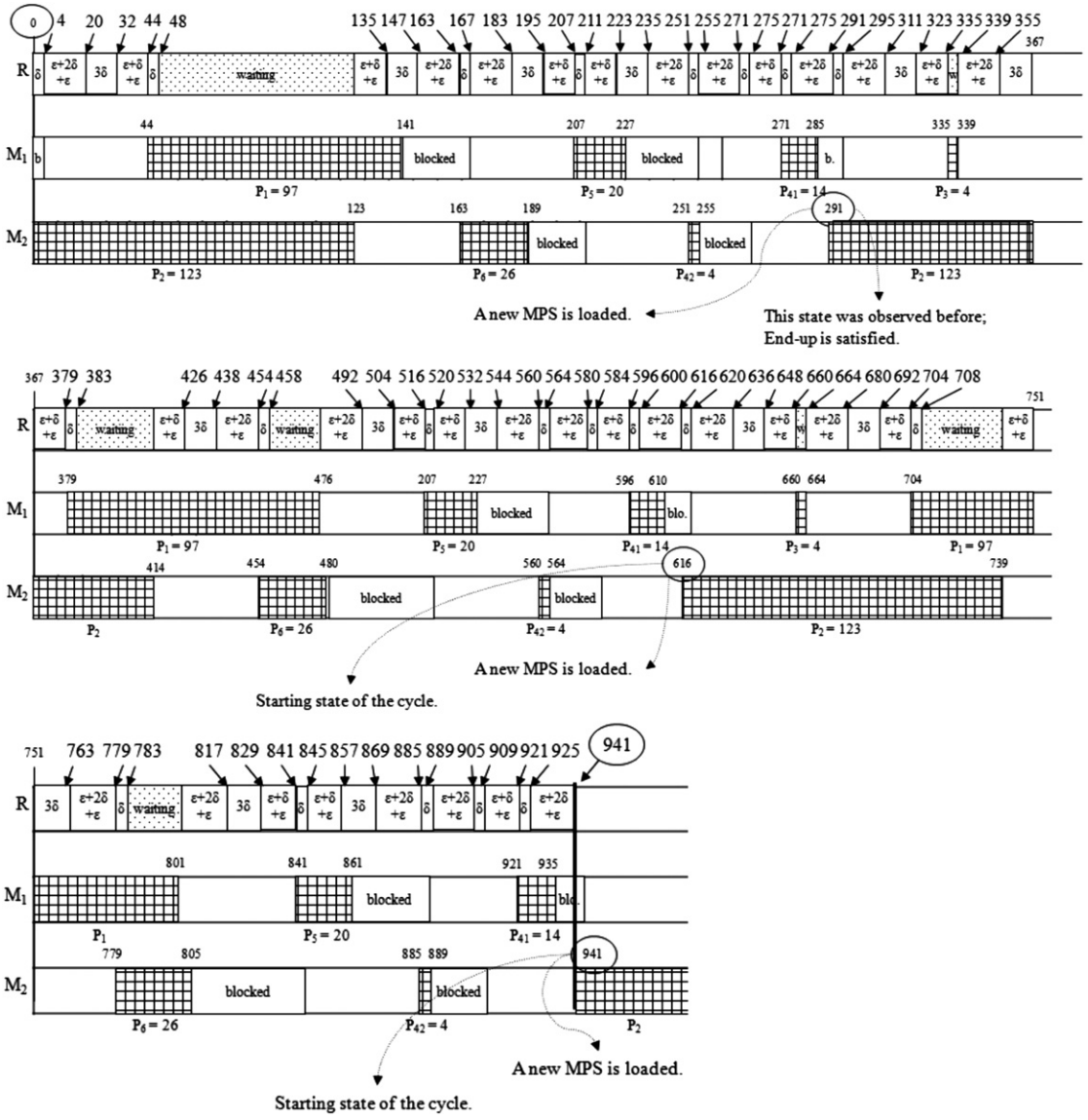

Fig. 12. Gantt-chart of Stage 2 on LPT for Example 2.

respectively. Considering Stage 1 of the example, total blocked time value of machine 1 is 34 and of machine 2 is 28. Since the difference between the machines equals to 6 and $(4\epsilon+4\delta)$ equals to 32, due to the first condition in Lemma 2, allocation is impossible. Therefore, we use the LPT solution of Example 2. As can be determined from Fig. 6, total blocked time value of machine 1 is 98 and of machine 2 is 52. Since the first one has more blocked time than the second, we set the first machine as machine x and the other one as machine y. As is mentioned, part to be loaded is determined according to the longest blocked time value on machine x, which is part 4 causing 44 units of blocked time with processing time of 18. Now we can start the calculations. B_d value equals to $|98-52|=46$ and the first condition in Lemma 3 is determined as follows:

$$P_x \le B_d-(4\epsilon+4\delta)=46-(4(4)+4(4))=14. \quad (20)$$

We have obtained our first condition as $P_x \le 14$.

For the second condition, we will consider $P_i-P_x-2\epsilon-4\delta$ and $2\epsilon+4\delta-P_x$ values. We solve this equation as follows:

$$\max\{0,P_i-P_x-2\epsilon-4\delta\}+\max\{0,2\epsilon+4\delta-P_x\} \le B_a-P_x, \quad (21)$$

$$\max\{0,18-P_x-2(4)-4(4)\}+\max\{0,2(4)+4(4)-P_x\} \le 44-P_x, \quad (22)$$

$$\max\{0,-6-P_x\}+\max\{0,24-P_x\} \le 44-P_x. \quad (23)$$

Since the two 'max' values in Eq. (23) give different results for different values of the allocated time, we will check for the following ranges on P_x value: For the first part;

- $\max\{0,-6-P_x\}$ gives the solution 0 for all P_x values.

For the second part:

- If $P_x \le 24$; then $0+24-P_x \le 44-P_x$, which gives $24 \le 44$ and since this is true, our second possibly valid interval is $P_x \le 24$.
- If $P_x \ge 25$; then $0+0 \le 44-P_x$, which gives $25 \le P_x \le 44$ as our third possibly valid interval.

We know that $P_x \le 14$ from the first equation of Lemma 1, combining this one with the newly obtained intervals, we get the following interval: $0 \le P_x \le 14$. After obtaining the interval, we will take the value closest to $2\epsilon+4\delta$ which is equal to 24, as P_x. We have obtained P_x value as 14. We can follow the next steps from the Gantt-chart given by Fig. 12.

As can be seen from the figure, we start the system as was determined by the LPT solution of assigning parts 1, 3 and 5 on machine 1, and 2, 6 on machine 2. For part 4, the one to be allocated, after loading it on its already assigned machine (machine 2), robot unloads the other machine (machine 1) and comes back to take the part for loading on the other machine. From now on, part 4's processing continues on this machine and robot goes to input buffer for the next assigned part. End-up policy used in this approach is also the same as previous; for this example, steady state is obtained at time 291. The time passed until that moment is $291-0=291$. Repeating the same movements once more, the system arrives to the same state at time 616, and 325 units of time passed until this point. On the third repetition, the same state is reached at time 941, which needed 325 units of time again. Since the last two results are the same, the algorithm ends with the solution of 325, where it was 339 in LPT.

References

Abdekhodaee, A.H., Wirth, A., Gan, H.S., 2006. Scheduling two parallel machines with a single server: the general case. Computers and Operations Research 33, 994–1009.

Akturk, M.S., Gultekin, H., Karasan, O.E., 2005. Robotic cell scheduling with operational flexibility. Discrete Applied Mathematics 145 (3), 334–348.

Aneja, Y.P., Kamoun, H., 1999. Scheduling of parts and robot activities in two-machine robotic cell. Computers & Operations Research 26, 297–312.

Blazewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., Weglarz, J., 2001. Scheduling Computer and Manufacturing Process, second ed. Springer.

Brauner, N., Finke, G., 2001. Optimal moves of the material handling system in a robotic cell. International Journal of Production Economics 74, 269–277.

Dawande, M., Geismar, N., Sethi, S.P., 2005. Dominance of cyclic solutions and challenges in the scheduling of robotic cells. SIAM Review 47 (4), 709–721.

Gilmore, P.C., Gomory, R.E., 1964. Sequencing in one state-variable machine: a solvable case of the traveling salesman problem. Operations Research 12, 655–679.

Hall, N.G., Kamoun, H., Sriskandarajah, C., 1997. Scheduling in robotic cells: classification, two and three machine cells. Operations Research 45 (3), 421–439.

Hall, N.G., Potts, C.N., Sriskandarajah, C., 2000. Parallel machine scheduling with a common server. Discrete Applied Mathematics 102, 223–243.

Kise, H., Shioyama, T., Ibaraki, T., 1993. Automated two machine flowshop scheduling: a solvable case. IIE Transactions 23, 80–87.

Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P., 1977. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1, 343–362.

Miller, C., Tucker, A., Zemlin, R., 1960. Integer programming formulation of traveling salesman problems. Journal of the ACM (JACM) 7 (4), 326–329.

Paul, H.J., Bierwirth, C., Kopfer, H., 2007. A heuristic scheduling procedure for multi-item hoist production lines. International Journal of Production Economics 105, 54–69.

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, 331–358.

Wen, C., Eksioglu, S.D., Greenwood, S., 2010. Crane scheduling in a shipbuilding environment. International Journal of Production Economics 124, 40–50.