Cyclic Scheduling of Parts and Robot Moves in m-machine Robotic Cells

Hakan Gultekin[a,*], Betul Coban[a], Vahid Eghbal Akhlaghi[b]

[a] *Department of Industrial Engineering, TOBB University of Economics and Technology, Ankara, Turkey*

[b] *Department of Industrial Engineering, Middle East Technical University, Ankara, Turkey*

Abstract

We consider a flow shop type manufacturing cell consisting of m machines and a material handling robot producing multiple parts. The robot transfers the parts between the machines and loads/unloads the machines. We consider the cyclic scheduling of the parts and the robot moves with the objective of maximizing the throughput rate. We develop a mixed integer linear programming formulation of the problem. The formulation is improved with several valid inequalities and reformulations of the constraints. We also develop a hybrid metaheuristic algorithm for this strongly NP-Hard problem. The algorithm is modified to handle both 1-unit and multi-unit robot cycles. Multi-threading is used to parallelize the algorithm in order to improve its efficiency. After calibrating the parameters of the heuristic algorithm, an extensive computational study is performed to evaluate its performance. The results of this study revealed that the developed heuristic provides near-optimal solutions in reasonable solution times. The effects of parallelization and the benefits of considering multi-unit cycles instead of 1-unit cycles are also quantified with this study.

Keywords: Robotic cell scheduling, Multiple parts, Mixed integer linear programming formulation, Hybrid metaheuristic, Throughput maximization

*Corresponding author

Email addresses: hgultekin@etu.edu.tr (Hakan Gultekin), bcoban@etu.edu.tr (Betul Coban), vahid.akhlaghi@metu.edu.tr (Vahid Eghbal Akhlaghi)

1. Introduction

The use of industrial robots increases continuously among the manufacturing firms. According to Robotic Industries Association, 14,583 robots valued at $817 million were ordered from North American robotics companies in the first half of 2016, an increase of two percent in units over the same period in 2015 (Robotics Online, 2016). The companies utilize industrial robots in order to increase productivity, safety, product quality, consistency, and flexibility. However, in order to get the maximum benefit from this high-cost investment, some complex operational problems must be solved. One of the most important operational problems is the sequencing and scheduling of the parts to be processed and the robot moves. In this study, we consider a flow shop type production system that consists of an input device denoted by M_0, a number of machines denoted by $M_1, \ldots, M_m$, an output device denoted by M_{m+1}, and a material handling robot as can be seen in Figure 1. Such a production system is called a robotic cell. Multiple parts with different processing times are to be processed in this cell. The problem is to determine the part sequence and the robot move sequence that jointly maximize the throughput or equivalently minimize the cycle time. The cycle time is defined as the long run average time to produce a single part.

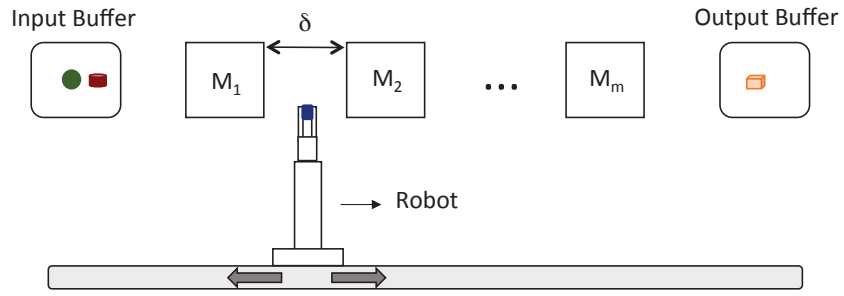


Figure 1: Robotic cell

In the literature mostly the makespan and the cycle time minimization objectives are considered for robotic cells. In the makespan minimization case, it is assumed that the production is not repetitive. All machines are initially idle, a specific number of parts are to be processed, and at the end, all machines become idle again. Soukhal and Martineau (2005) developed an integer programming model and a Genetic Algorithm (GA) based heuristic for this problem. Carlier et al. (2010) proposed an exact branch-and-bound

algorithm and developed a GA. Elmi and Topaloglu (2014) considered a hybrid flow shop type robotic cell in which there are different speed parallel machines at each stage. They developed an integer programming formulation and proposed a Simulated Annealing (SA) based algorithm. Liu and Kozan (2016) considered the job shop version of the problem and developed a hybrid metaheuristic algorithm.

In the cycle time minimization case, it is assumed that the same set of parts are to be processed indefinitely. Therefore, the system is not required to start with an empty state in which all machines are idle. The state of the system can be identified with the position of the robot and the status of all machines (whether they are idle or loaded with a part). In order to satisfy repetitiveness, the final state of a cycle must be identical to its initial state. Such cyclic schedules are used extensively especially in mass production where the same set of parts are produced in large volumes. In a robotic cell, cyclic production refers to the repetition of the same sequence of robot moves indefinitely. Cyclic schedules are easy to implement and control for real life applications. Furthermore, Geismar et al. (2005) proved that such schedules are optimal for robotic flow shops producing identical parts. Sethi et al. (1992) defined an *n-unit* cycle as a sequence of robot activities in which the system returns to the same state after producing n parts. In these cycles, each machine is loaded and unloaded exactly n times and n parts are produced at the end.

Most studies on robotic cell scheduling problems assume the parts to be identical. With this assumption, the part sequencing problem vanishes and the robot move sequencing remains as the only problem. Sethi et al. (1992) proved that 1-unit cycles are optimal for two-machine cells producing identical parts. Crama and van de Klundert (1997) extended this result for three-machine cells. However, for m-machine cells where $m \geq 4$, Brauner and Finke (2001) showed that a multi-unit cycle can lead to a cycle time that is smaller than the smallest 1-unit cycle time. There are several studies that consider multi-unit cycles and develop exact or heuristic solution procedures for the identical parts case (see e.g. Kats et al., 1999; Che et al., 2011; Zhou et al., 2012; Li and Fung, 2014; Elmi and Topaloglu, 2016).

In the multiple part-types case, the repetitive production of a *Minimal Part Set (MPS)* is considered. An MPS is the smallest possible set of parts having the same proportions as the production target. For example, if the demands for three products for some fixed

period of time are given as 400 for product A, 250 for product B, and 150 for product C, then the MPS consists of 8 product A's, 5 product B's, and 3 product C's that makes a total of 16 products. Given an MPS of n parts, an *MPS cycle* is equivalent to an n-unit robot move cycle. The cycle time can be defined as the total time to produce all n parts in the MPS. We utilize this definition of the cycle time throughout this study,. The per unit cycle time can be found easily by dividing this total time by n. A *Concatenated Robot Move (CRM)* cycle is a special class of MPS cycles in which the same 1-unit cycle is repeated n times (Sriskandarajah et al., 2004). The reason for considering CRM cycles lies in the easiness of implementation and control of these cycles. In most studies in the literature, the robot move sequence is fixed to a specific CRM cycle and the corresponding optimal part sequence is determined accordingly (Dawande et al., 2007). Sethi et al. (1992) solved the part sequencing problem for a given CRM cycle in a two-machine cell. Hall et al. (1998) proved that finding the optimal part sequence in two of the six possible CRM cycles in a three-machine robotic cell is strongly NP-Hard. Sriskandarajah et al. (1998) generalized these results for the m-machine cells and categorized the part sequencing problem for the CRM cycles into four categories depending on their complexity status. Kamalabadi et al. (2008) considered the part sequencing problem when the CRM cycle is fixed in a three-machine robotic cell and proposed a Particle Swarm Optimization (PSO) method. Abdulkader et al. (2013) considered the same problem in a four-machine robotic cell and proposed a GA. On the other hand, Hall et al. (1997) proved that the optimal cycle is not generally a CRM cycle even for the two-machine cells. They developed an algorithm with $O(n^4)$ complexity to determine the optimal MPS cycle for the two-machine case. Aneja and Kamoun (1999) improved the time complexity to $O(nlogn)$ by formulating the problem as a special case of the *Traveling Salesman Problem (TSP)*. However, the problem is strongly NP-Hard for the m-machine robotic cells when $m \geq 3$ and the robot sequence is not fixed (Hall et al., 1998).

Zahrouni and Kamoun (2012) developed a constructive heuristic inspired by the NEH algorithm of Nawaz et al. (1983) for the three-machine case. Batur et al. (2012) formulated the two-machine problem in which the processing times are also decision variables as a variation of the TSP. Since the model size increases drastically with respect to m, extending this formulation to the m-machine case is not practical. Fazel Zarandi et al.

(2013) developed a branch and bound algorithm and proposed an SA algorithm for the two-machine case that includes sequence-dependent setup times. Che et al. (2010) considered the cyclic hoist scheduling with multiple parts, fixed processing times, and no-wait constraints. They developed an MILP formulation which makes use of the no-wait constraints and developed a dynamic branch and bound procedure.

In this study, we consider both the CRM and MPS cycles and develop an MILP formulation of the general m-machine case that determines the part sequence and the robot move sequence simultaneously. As far as the authors know, this is the first formulation to solve this general problem. We develop a lower bound and several valid inequalities and utilize these in the formulation. However, since this problem is strongly NP-Hard (Hall et al., 1997), we develop a Parallel Hybrid Metaheuristic algorithm to find near-optimal solutions in reasonable CPU times for the MPS case. We modify this algorithm to handle the CRM case as well. In this approach, a parallel GA determines the part sequence. For a given part sequence, a Tabu Search (TS) algorithm determines the corresponding best robot move cycle and calculates the resulting cycle time. The performance of the heuristic is analyzed through an extensive computational study by comparing its results with the MILP. Furthermore, in order to quantify the benefits of utilizing multi-unit cycles in terms of the throughput rate, 1-unit and multi-unit cycles are compared with each other. This is important since there is a tradeoff between the throughput rate and the easiness of implementing and controlling the cycle. This quantification helps decision makers to make a choice between the CRM cycles and more complex MPS cycles.

Table 1 lists the closely related studies and provides the position of the current study with respect to them. The studies that consider multiple part-types either consider the makespan or the cycle time objectives. Studies considering cycle time minimization usually assume the robot move sequence (RMS) or the part sequence (PS) to be given and try to determine the other one. Furthermore, usually the problem is limited to the CRM cycles which are not guaranteed to be optimal. Although a few studies examine the problem of determining the best multi-unit cycle, these studies consider a small number of machines (e.g. 2 or 3). According to this table, the current study is the first one to consider the cyclic scheduling of both the robot moves and the parts in the general m-machine case. Both the CRM and the MPS cycles are considered and their performances

Table 1: Multiple part type robotic cell scheduling literature classification

Study	Cyclic Solution	multi-unit	m-machine	RMS	PS	Main Contribution
Sethi et al. (1992)	✓				✓	Polynomial time algorithm
Hall et al. (1997)	✓	✓		✓	✓	$O(n^4)$ time algorithm
Hall et al. (1998)	✓			✓		Complexity analysis of part sequencing for CRM cycles
Aneja and Kamoun (1999)	✓	✓		✓	✓	$O(nlogn)$ time algorithm
Soukhal and Martineau (2005)			✓	✓	✓	MILP formulation and a GA
Kamalabadi et al. (2008)	✓			✓		Petri nets based mathematical model and a PSO
Carlier et al. (2010)			✓	✓	✓	Branch & bound algorithm for PS and a GA for RMS
Zahrouni and Kamoun (2012)	✓	✓		✓	✓	NEH based constructive heuristic
Batur et al. (2012)	✓	✓		✓	✓	MILP formulation and a heuristic algorithm
Abdulkader et al. (2013)	✓				✓	GA for PS for each of the 24 CRM cycles
Fazel Zarandi et al. (2013)	✓			✓	✓	MILP formulation and a SA algorithm
Elmi and Topaloglu (2014)			✓	✓	✓	MILP formulation and a SA algorithm
Current Study	✓	✓	✓	✓	✓	**MILP formulation, parallel hybrid metaheuristic algorithm, and comparison of CRM and MPS cycles**

RMS: Robot Move Sequence, **PS:** Part Sequence

are compared with each other.

The remainder of the paper is organized as follows. The notation, problem definition, and mathematical programming formulations are presented in the next section. Section 3 presents the proposed hybrid metaheuristic algorithm. In Section 4 the methodology and the results of the computational tests are summarized. Section 5 concludes the study.

2. Problem Definition

In this section, we provide the formal definition of the problem and develop an MILP formulation. We also develop a lower bound for the cycle time and determine several valid

inequalities and utilize these of the model. We evaluate their effects on the solution time of the model using several test problems. According to the flow shop assumption, each part to be processed goes through each machine in the same sequence; $M_0, M_1, \ldots, M_m, M_{m+1}$. In the considered system, the loading/unloading of the machines and part transfers between the machines are performed by the robot. Since the robot can hold only one part at a time and there are no buffers between the machines, it is possible to represent robot move cycles with a robot activity definition. Crama and van de Klundert (1997) defined A_i, $i = \{0, 1, 2, \ldots, m\}$ as the robot activity in which the robot unloads a part from M_i, travels from M_i to M_{i+1}, and loads this part to M_{i+1}. Each loading/unloading time is assumed to be equal and denoted by ε. After loading a part on a machine, the robot either waits in front of that machine or moves to another machine to take a new part.

Let p_{ij} denote the processing time of part j on M_i. In this study, we consider free pickup criterion in which a part can stay on the machine indefinitely after its processing is completed. The machines in the system are equidistant and the robot's travel time between adjacent machines (M_{i-1} and M_i, $i = 1, \ldots, m+1$) is denoted by δ. This travel time is additive. That is, the travel time between any two machines M_i, M_j, $0 \leq i, j \leq m+1$ is $|i-j|\delta$. The following example is useful in understanding the problem and the operation of the cell.

Example 1. Let's consider a three-machine cell where there are $n = 4$ parts in the MPS. Loading/unloading and travel times are given as $\varepsilon = 1$ and $\delta = 1$. The processing times are provided in Table 2. The optimal solution is found by the MILP model developed later in this section. The optimal part sequence is $\sigma_{PS} = \{4, 1, 2, 3\}$ and the robot activity sequence (σ_{RMS}) together with their starting times are provided in Table 3. The corresponding Gantt chart is depicted in Figure 2.

Table 2: Processing times for Example 1

m\n	**1**	**2**	**3**	**4**
1	13	16	22	12
2	7	11	9	5
3	8	4	14	13

Table 3: The RMS and activity start times for Example 1

σ_{RMS}	A_0	A_2	A_1	A_0	A_3	A_2	A_1	A_0	A_3	A_2	A_1	A_0	A_3	A_2	A_3	A_1
T	0	7	15	20	26	31	36	41	47	52	60	65	70	77	84	90

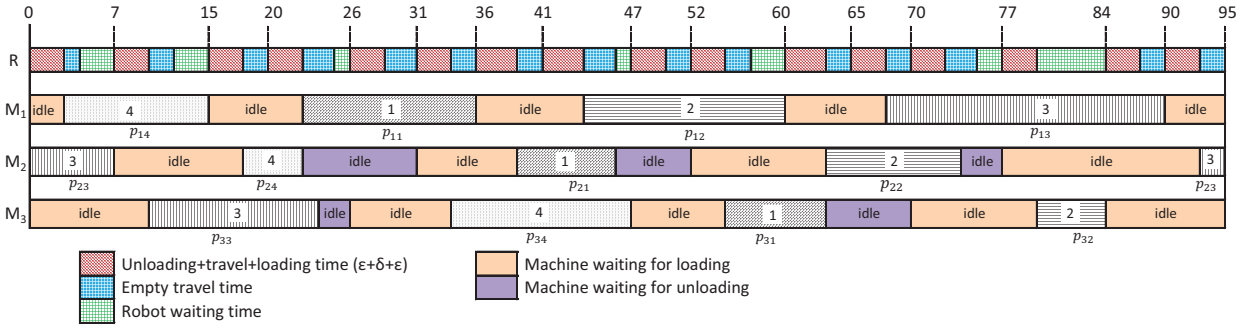


Figure 2: Gantt chart for Example 1

In an n-unit cycle each activity A_i, $i = (0, 1, 2, \ldots, m)$ must be performed exactly n times. The developed mathematical model determines the sequence of these activities by assigning them to positions. Since there are a total of $(m + 1)$ activities and since each of them are repeated n times, the total number of positions is $n(m + 1)$.Crama and van de Klundert (1997) defined the feasibility conditions for robot move cycles. According to this, the robot cannot be instructed to load an occupied machine and cannot be instructed to unload an unoccupied machine. For 1-unit cycles, all possible permutations of the activities lead to a feasible cycle. However, this is not valid in an n-unit cycle. Crama and van de Klundert (1997) defined the necessary and sufficient conditions for an activity sequence to represent a feasible cycle. According to this, between two consecutive occurrences of A_i, $i = (0, 1, \ldots, m - 1)$, there must be exactly one A_{i+1} and between two consecutive occurrences of A_i, $i = (1, 2, \ldots, m)$, there must be exactly one A_{i-1}. These conditions will be used in order to generate feasible activity sequences in the following model. The model calculates the starting time of each activity depending on its position in the activity sequence.

In the following formulation, B is used to denote a very large number. This number must at least be larger than the optimal cycle time value. Therefore, B can be selected as the cycle time of any feasible cycle with the corresponding part sequence. The CRM

cycle that repeats the $A_0A_1 \dots A_m$ sequence n times is a very good candidate for this purpose. This is because, the cycle time formula of this cycle, which can be calculated as $2n(m+1)(\varepsilon+\delta) + \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}$, does not depend on the part sequence.

2.1. Mixed Integer Linear Programming Formulation

We use the following sets and parameters in the MILP formulation:

$\mathcal{N} = \{1, 2, \dots, n\}$: The set of parts. Since each activity A_i is repeated n times in an MPS cycle, this set is also used to represent these repetitions,

$\mathcal{M} = \{M_0, M_1, \dots, M_{m+1}\}$: The set of machines. M_0 is the input buffer and M_{m+1} is the output buffer,

$\mathcal{A} = \{0, 1, \dots m\}$: The set of activity indices, $(A_0, A_1, \dots, A_m)$,

$\mathcal{P} = \{1, 2, \dots, n(m+1)\}$: The set of positions to represent an n-unit cycle,

p_{ij} : The processing time of part $j \in \mathcal{N}$ on machine $i \in \mathcal{M} \setminus \{M_0, M_{m+1}\}$,

ε : Loading/unloading time,

δ : Traveling time of the robot between adjacent machines,

B : A very large number.

We define the following decision variables:

$$x_{ilk} = \begin{cases} 1, & \text{If repetition } l \in \mathcal{N} \text{ of activity } i \in \mathcal{A} \text{ is assigned to position } k \in \mathcal{P}, \\ 0, & \text{Otherwise.} \end{cases}$$

$$z_{ilj} = \begin{cases} 1, & \text{If repetition } l \in \mathcal{N} \text{ of activity } i \in \mathcal{A} \text{ belongs to part } j \in \mathcal{N}, \\ 0, & \text{Otherwise.} \end{cases}$$

We also introduce y_{jlok} in order to linearize the multiplication $x_{ilk}.z_{ilj}$:

$$y_{ilkj} = \begin{cases} 1, & \text{If repetition } l \in \mathcal{N} \text{ of activity } i \in \mathcal{A} \text{ belongs to part } j \in \mathcal{N} \text{ and it is} \\ & \text{assigned to position } k \in \mathcal{P}, \\ 0, & \text{Otherwise.} \end{cases}$$

In order to calculate the cycle time, the starting time of individual activities must be known. In order to calculate these we define the following decision variables:

T_k : The starting time of the activity that is assigned to position $k \in \mathcal{P}$,

CT : The cycle time.

The assignment of parts to activities and activities to positions are satisfied by the following constraints:

$$\sum_{k \in \mathcal{P}} x_{ilk} = 1 \qquad \forall i \in \mathcal{A}, \forall l \in \mathcal{N} \tag{1}$$

$$\sum_{l \in \mathcal{N}} \sum_{i \in \mathcal{A}} x_{ilk} = 1 \qquad \forall k \in \mathcal{P} \tag{2}$$

$$\sum_{l \in \mathcal{N}} z_{ilj} = 1 \qquad \forall i \in \mathcal{A}, \forall j \in \mathcal{N} \tag{3}$$

$$\sum_{j \in \mathcal{N}} z_{ilj} = 1 \qquad \forall i \in \mathcal{A}, \forall l \in \mathcal{N} \tag{4}$$

$$y_{ilkj} \geq x_{ilk} + z_{ilj} - 1 \qquad \forall i \in \mathcal{A}, \forall j, l \in \mathcal{N}, \forall k \in \mathcal{P} \tag{5}$$

$$x_{ilk} \geq y_{ilkj} \qquad \forall i \in \mathcal{A}, \forall j, l \in \mathcal{N}, \forall k \in \mathcal{P} \tag{6}$$

$$z_{ilj} \geq y_{ilkj} \qquad \forall i \in \mathcal{A}, \forall j, l \in \mathcal{N}, \forall k \in \mathcal{P} \tag{7}$$

Constraint (1) ensures that each repetition of each activity is assigned to one position whereas Constraint (2) ensures that a single repetition of a single activity is assigned to each position. For each part, each activity must be utilized once as satisfied by Constraint (3) and each repetition of each activity must belong to a single part as imposed by Constraint (4). $x_{ilk}.z_{ilj}$ multiplication is linearized in Constraints (5), (6), and (7). The following set of constraints calculates the starting times of the activities.

$$T_k \geq T_h + 2\varepsilon + \delta + \sum_{l \in \mathcal{N}} \sum_{j \in \mathcal{N}} y_{ilkj}.p_{ij} + B\left(\sum_{l \in \mathcal{N}} x_{ilk} + \sum_{l \in \mathcal{N}} x_{(i-1)lh} - 2\right) \qquad \forall k, h \in \mathcal{P} : k > h,\ \forall i \in \mathcal{A} \setminus \{0\} \tag{8}$$

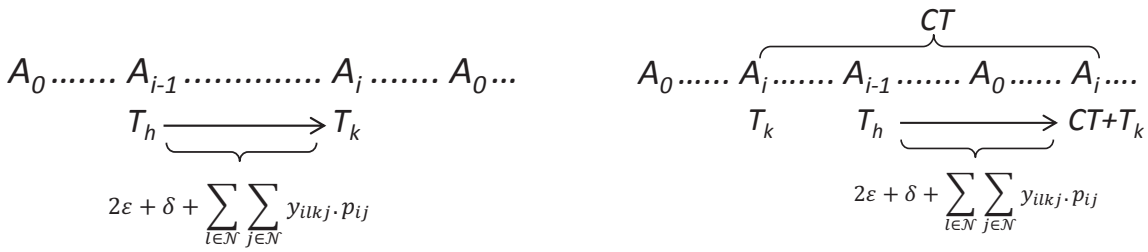
$$CT + T_k \geq T_h + 2\varepsilon + \delta + \sum_{l \in \mathcal{N}} \sum_{j \in \mathcal{N}} y_{ilkj}.p_{ij} + B\left(\sum_{l \in \mathcal{N}} x_{ilk} + \sum_{l \in \mathcal{N}} x_{(i-1)lh} - 2\right) \qquad \forall k, h \in \mathcal{P} : k < h,\ \forall i \in \mathcal{A} \setminus \{0\} \tag{9}$$

$$T_{k+1} \geq T_k + 2\varepsilon + \delta + \delta|g - (i+1)| + B\left(\sum_{l \in \mathcal{N}} x_{ilk} + \sum_{l \in \mathcal{N}} x_{gl(k+1)} - 2\right) \qquad \forall i, g \in \mathcal{A} : i \neq g,\ \forall k \in \mathcal{P} \setminus \{n(m+1)\} \tag{10}$$

$$CT \geq T_{n(m+1)} + 2\varepsilon + \sum_{i \in \mathcal{A}} \sum_{l \in \mathcal{N}} (i+2)\delta.x_{il(n(m+1))} \tag{11}$$

$$T_1 = 0 \tag{12}$$

If A_{i-1} is assigned to position h and A_i is assigned to position k where $k > h$ then, the starting time of A_i (denoted by T_k in this case) must be later than the completion of the processing of the part on this machine as depicted in Figure 3a. Activity A_{i-1} starts at time T_h. The robot unloads M_{i-1}, moves to M_i, and loads it, which takes a total of $2\varepsilon + \delta$ amount of time. The processing of the part on M_i starts at this time. This relation between T_k and T_h is formulated in Constraint (8). On the other hand, if A_{i-1} appears later than A_i in the activity sequence ($k < h$), this means that in the initial state of the cycle, M_i is loaded. When the robot arrives in front of this machine to unload it, in order to guarantee that the processing is completed, the starting time of A_i (T_k) plus the cycle time (CT) must be larger than the completion time of the part on M_i as depicted in Figure 3b. This is satisfied by Constraint (9).



(a) $k > h$ (b) $k < h$

Figure 3: Relation between starting times of activities

Constraint (10) determines the relation between the starting times of activities that are assigned to consecutive positions. If A_i is in position k and A_g is in position $k+1$, then the starting time of A_g (T_{k+1} in this case) must be later than the completion time of A_i plus the robot travel time from M_i to M_g. Starting at time T_k, the robot unloads M_i, moves to M_{i+1}, and loads it after $2\varepsilon + \delta$ amount of time. Then the robot moves from M_{i+1} to M_g in $|g - (i+1)|\delta$ time. Constraint (11) calculates the cycle time, which must be greater than the completion time of the last activity in the sequence plus the necessary time for the robot to return back to the input buffer. Constraint (12) sets the starting time of the first activity. The following set of constraints satisfies the feasibility

conditions for the activity sequence.

$$\sum_{k\in\mathcal{P}} k.x_{ilk} \leq \sum_{k\in\mathcal{P}} k.x_{irk} \qquad \forall i \in \mathcal{A},\ \forall l, r \in \mathcal{N} : l < r \tag{13}$$

$$\sum_{r\in\mathcal{N}} \sum_{s=k+1}^{h-1} x_{(i-1)rs} \geq 1 - B(2 - x_{ilk} - x_{i(l+1)h}) \qquad \forall k, h \in \mathcal{P} : k < h, \forall i \in \mathcal{A} \setminus \{0\},\ \forall l \in \mathcal{N} \setminus \{n\} \tag{14}$$

$$\sum_{r\in\mathcal{N}} \sum_{s=k+1}^{h-1} x_{(i+1)rs} \geq 1 - B(2 - x_{ilk} - x_{i(l+1)h}) \qquad \forall k, h \in \mathcal{P} : k < h, \forall i \in \mathcal{A} \setminus \{m\}, \forall l \in \mathcal{N} \setminus \{n\} \tag{15}$$

$$\sum_{j\in\mathcal{N}} y_{011j} = 1 \tag{16}$$

$$\sum_{l\in\mathcal{N}} \sum_{s=k+1}^{h-1} y_{(i+1)lsj} \geq 1 - B\left(2 - \sum_{l\in\mathcal{N}} y_{ilkj} - \sum_{l\in\mathcal{N}} \sum_{t\in\mathcal{N}} y_{(i+1)lht}\right) \qquad \forall i \in \mathcal{A} \setminus \{m\}, \forall j \in \mathcal{N}, \forall k, h \in \mathcal{P} : k < h \tag{17}$$

$$x_{ilk}, z_{ilj}, y_{ilkj} \in \{0, 1\} \qquad \forall i \in \mathcal{A}, \forall j, l \in \mathcal{N}, \forall k \in \mathcal{P} \tag{18}$$

$$T_k, CT \geq 0 \qquad \forall k \in \mathcal{P} \tag{19}$$

Constraint (13) satisfies that the earlier repetition of an activity must be assigned to an earlier position. In order an activity sequence to be feasible, between any two repetitions of A_i, $i \in \mathcal{A} \setminus \{0\}$, there must be one A_{i-1}.This is satisfied by Constraint (14). Similarly, between any two repetitions of A_i, $i \in \mathcal{A} \setminus \{m\}$, there must be one A_{i+1} as satisfied by Constraint (15). We assume that all cycles start with activity A_0 without loss of generality as satisfied by Constraint (16). If an A_i activity belongs to part j, then the first A_{i+1} following this A_i must also belong to part j. This is satisfied by Constraint (17).

Using these constraints the problem named as MILP1 can be formulated as follows:

MILP1: Minimize CT

s.t. Constraints $(1) - (19)$

2.2. Lower Bound and Valid Inequalities

In this section we first derive a lower bound on the cycle time. We also determine several valid inequalities for MILP1. We will utilize the lower bound and the valid inequalities into MILP1 to reduce its solution time. Several lower bounds are developed for

different cell configurations (see e.g. Crama and van de Klundert (1997), Gultekin et al. (2008), and Gultekin et al. (2009)). In the following, we derive a lower bound for the current cell configuration.

Theorem 1. *The optimal cycle time of MILP1 can not be less than LB where,*

$$LB = \max\left\{2n(m+1)(\epsilon+\delta) + \sum_{j=1}^{n}\sum_{i=1}^{m}\min\{p_{ij},\delta\}, 4n(\epsilon+\delta) + \max_{i}\left\{\sum_{j=1}^{n} p_{ij}\right\}\right\} \quad (20)$$

Proof. The lower bound is given as the maximum of two terms. The first term calculates the total time that the robot must perform in order to complete a cycle. Each part to be processed must start from the input buffer, visit all machines and must be dropped to the output buffer. Which means that the robot must travel from the input buffer to the output buffer and return back to the input buffer for each part. In an m-machine cell, the distance between the input and the output buffers is $(m+1)\delta$. Therefore, the total travel time to complete n parts must be at least $2n(m+1)\delta$. Besides this, each part must be taken from the input buffer, visit each machine, and dropped to the output buffer. This makes a total of $2n(m+1)\epsilon$ loading/unloading time for all parts. After loading a part to a machine, the robot either waits in front of the machine to complete the processing or moves to another machine to make some other activities. If it moves to another machine the minimum time before starting a new activity is δ. Therefore, the minimum of these two values for each part and each machine this must be included in the lower bound function. This makes $\sum_{j=1}^{n}\sum_{i=1}^{m}\min\{p_{ij},\delta\}$. Summing up this value with the total loading/unloading and travel times, we get the first term of the lower bound function.

The cycle time can also be defined as the time between the starting times of the same activities corresponding to the same part in the two consecutive repetitions of the MPS sequence. The second term of the max function in Equation (20) utilizes this definition. When the robot starts unloading a machine, M_i, it must complete the following minimal set of moves before performing the same activity in the next repetition of the MPS cycle: Unload M_i (ϵ), move to M_{i+1} (δ), load this machine (ϵ), move to M_{i-1}, (2δ), unload this machine (ϵ), move to M_i (δ), load this machine (ϵ), wait for the processing to be completed, (p_{ij}). This set of moves must be repeated for each part in the MPS. Therefore,

the total time is $4n(\epsilon+\delta)+\sum_{j=1}^{n} p_{ij}$. On the other hand, a different lower bound appears by considering each machine separately. The best lower bound is found by taking the maximum of these that leads to the second term of the max function. $\square$

MILP1 is the complete formulation of the problem. However, we can include the following two constraints into MILP1 in order to increase its performance.

$$\sum_{l\in\mathcal{N}}\sum_{k\in\mathcal{P}} x_{ilk} = n \qquad \forall i \in \mathcal{A} \tag{21}$$

$$\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{P}} k.y_{ilkj} \leq \sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{P}} k.y_{irkj} \qquad \forall i \in \mathcal{A},\ \forall l,r \in \mathcal{N} : l < r \tag{22}$$

Equation (21) states that each activity will be used n times in the activity sequence of any cycle. Equation (22) is similar to Constraint (13) and satisfies that the earlier repetition of an activity is assigned to an earlier position. Besides these additional constraints, we can also reformulate Constraint (10) as follows:

$$T_{k+1} \geq T_k + 2\varepsilon + \delta + \delta \left| \sum_{i\in\mathcal{A}}\sum_{l\in\mathcal{N}} i.x_{il(k+1)} - \left(\sum_{i\in\mathcal{A}}\sum_{l\in\mathcal{N}} i.x_{ilk} + 1 \right) \right| \quad \forall k \in \mathcal{P}\backslash\{n(m+1)\} \tag{23}$$

This constraint satisfies the necessary relation between T_{k+1} and T_k as in Constraint (10) without using a big number B which may improve the relaxation bounds. The absolute value calculates the distance between the machines corresponding to the activities that are assigned to positions k and $k+1$. In order to linearize this constraint, we define two new positive decision variables, u_k and v_k and replace the constraint with the following two constraints. Since the cycle time is being minimized, and since the cycle time is large than all starting times of the activities, this linearization works correctly.

$$T_{k+1} \geq T_k + 2\varepsilon + \delta + \delta(u_k + v_k) \qquad \forall k \in \mathcal{P} \setminus \{n(m+1) \tag{24}$$

$$u_k - v_k = \sum_{i\in\mathcal{A}}\sum_{l\in\mathcal{N}} i.x_{il(k+1)} - \left(\sum_{i\in\mathcal{A}}\sum_{l\in\mathcal{N}} i.x_{ilk} + 1 \right) \quad \forall k \in \mathcal{P} \setminus \{n(m+1) \tag{25}$$

As a consequence, we can formulate MILP2 as follows:

MILP2: Minimize CT

s.t. Constraints $(1)-(9),(11)-(22),(24),(25)$

Note that, in MILP2, the lower bound in Equation (20) is written as a constraint as $CT \geq LB$. In order to test the effects of these valid inequalities and reformulations we

modeled both formulations in General Algebraic Modeling System (GAMS) and solved using the CPLEX 12.6.2 solver with a time limit of 3600s. The tests are performed on a machine with Intel Xeon E5645 2.40 GHz CPU (2×6 cores), 18 GB of RAM under a 64bit operating system. We used a total of 24 test problems, which differ in terms of the number of parts, the number of machines, as well as the loading/unloading time. The processing times are generated randomly. The problem complexity depends on the magnitude of δ with respect to the processing times. If the δ value is too small, then one specific cycle usually becomes optimal, whereas when it is too large, another specific cycle becomes optimal. In order to avoid these instances, δ values are selected accordingly for each problem.

Table 4: Comparison of MILP1 and MILP2

	MILP1		MILP2	
(n, m, δ)	Gap (%)	CPU Time (s)	Gap (%)	CPU Time (s)
$(3, 3, 50)$	0	2.01	0	2.28
$(3, 3, 100)$	0	1.92	0	1.50
$(4, 3, 50)$	0	38.81	0	29.00
$(4, 3, 100)$	0	80.73	0	19.64
$(5, 3, 30)$	0	1039.98	0	505.97
$(5, 3, 75)$	0	1794.12	0	583.42
$(6, 3, 50)$	120.46	TL	9.69	TL
$(6, 3, 100)$	138.59	TL	16.00	TL
$(3, 4, 30)$	0	61.70	0	40.16
$(3, 4, 60)$	0	96.73	0	58.61
$(4, 4, 30)$	43.43	TL	0	2678.62
$(4, 4, 60)$	98.21	TL	8.75	TL
$(5, 4, 50)$	130.35	TL	11.22	TL
$(5, 4, 100)$	212.98	TL	24.94	TL
$(6, 4, 40)$	NA	TL	NA	TL
$(6, 4, 80)$	NA	TL	NA	TL
$(3, 5, 30)$	54.63	TL	0	708.45
$(3, 5, 60)$	80.43	TL	0	1150.11
$(4, 5, 40)$	101.18	TL	16.69	TL
$(4, 5, 80)$	170.41	TL	20.22	TL
$(5, 5, 40)$	NA	TL	NA	TL
$(5, 5, 75)$	NA	TL	NA	TL
$(6, 5, 40)$	NA	TL	NA	TL
$(6, 5, 75)$	NA	TL	NA	TL

* TL: Time Limit, NA: Not Available

All test results are presented in Table 4. TL represents Time Limit in this table. In some cases, the formulations cannot find any integer solution in a one-hour time limit. These situations are denoted with NA. As it can be seen, MILP1 found the optimal solution in 8 of the 24 instances (gap=0), whereas MILP2 found in 11 of the instances.

The average and maximum percent gaps of all instances for MILP1 are 63.93 and 212.98, respectively. These are 5.97 and 20.22 for MILP2. The CPU times are also better for MILP2 nearly in all instances. Therefore, we conclude that MILP2 has a better performance and we will use this formulation in the rest of this study. However, the solution time of MILP2 increases drastically with the problem size and there are instances for which MILP2 could not find even an integer feasible solution in one-hour time limit. As a consequence, it is necessary to develop a heuristic algorithm.

3. Hybrid Metaheuristic Algorithm

In this section, we develop a hybrid meta-heuristic algorithm that solves both the part sequencing and the robot move sequencing problems. The heuristic is developed to generate MPS cycles. We later modify it to generate CRM cycles as well. GA is a widely used heuristic algorithm and is very suitable for the part sequencing problem of the current study. However, it is not very suitable for the robot move sequencing problem, which has a number of feasibility conditions as explained before. Therefore, the classical crossover operators of the GA may not yield feasible offsprings. A TS algorithm with a carefully defined neighborhood is more suitable for the robot move sequencing problem. Therefore we combined these two algorithms. The part sequence is determined by the GA at the top level. Then for each individual in the population (i.e., the part sequence) the corresponding robot move sequence is determined by a TS algorithm and its fitness value (the cycle time) is calculated with an iterative procedure. Furthermore, in order to reduce the solution time we parallelized the GA. That is, after the individuals of a population (part sequence) are determined, the robot move sequence is determined and their cycle times are calculated by distributing the individuals to different cores of the CPU. The flowchart of the developed algorithm can be seen in Figure 4.

The details of the developed GA and TS are explained in the following two sections, the modification of the hybrid algorithm to handle CRM cycles is explained in Section 3.3, and the parameters of the heuristic are calibrated with a computational study as explained in Section 3.4.

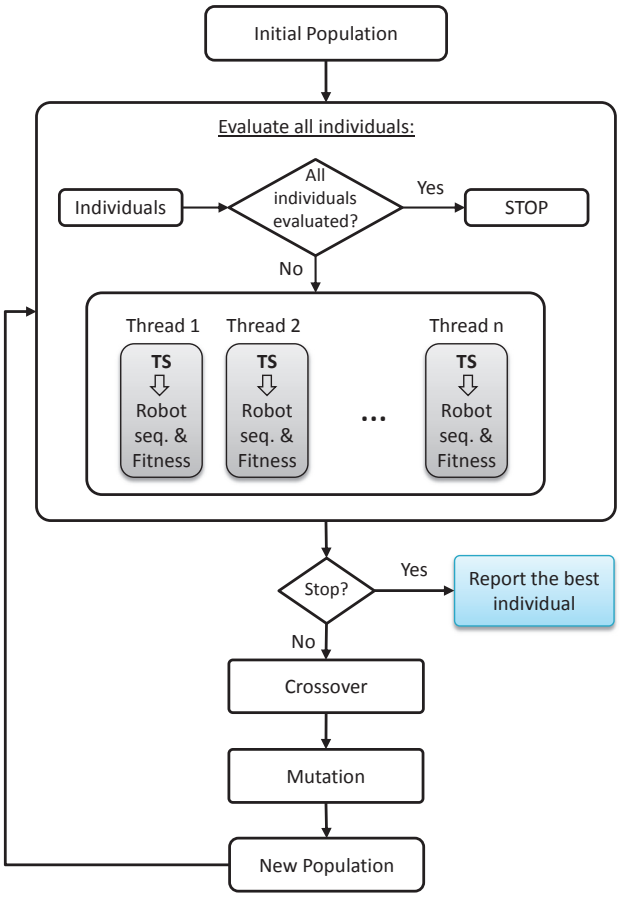


Figure 4: Flow chart of the parallel hybrid metaheuristic algorithm

3.1. Part Sequencing

The main features of the GA that is used for sequencing the parts are as follows:

Solution encoding: A feasible solution for this problem is a permutation of parts. $\sigma_{PS} = \{j_1, j_2, \ldots, j_n\}$ is used to represent a chromosome. Since the production is cyclic and the same MPS is repeated indefinitely, the total number of permutations is $(n-1)!$.

Population size: Since the number of distinct permutations depend on n, instead of utilizing a constant population size for all problem instances, we determined population size as a function of the number of parts, $\beta \times n$. The value of the β parameter is determined by computational tests.

Fitness computation: Given an individual, its fitness is calculated by a TS algorithm, which will be explained in Section 3.2. The TS algorithm first determines the corresponding robot move sequence. Then the cycle time is calculated by an iterative procedure.

Elite solutions: A number of elite individuals are moved directly to the next generation.

By this way, "fittest individuals" are maintained through different generations. Elite solutions are determined depending on their fitness values. The number of elite solutions to be moved to the next generation are determined as a percentage, θ, of the population size. The value of this parameter is determined by computational tests. Elite solutions are also considered as potential parents for crossover.

Crossover: Roulette wheel selection method is used to implement crossover. The probability of selecting an individual for crossover, ρ_j, is based on its fitness values and calculated as follows:

$$\rho_j = \frac{1/CT_j}{\sum_{j=1}^{\beta.n}(1/CT_j)},$$

where CT_j is the cycle time of the j^{th} individual. According to this formula, if a solution has a better fitness value, it has a higher probability of selection.

Two parents are selected from the population according to their selection probabilities. Similar Job Order Crossover method of Ruiz and Maroto (2006) is used as the crossover operator. In this method, a random cutting point is determined. The job sequence of the first (second) offspring until this cutting point is inherited directly from the first (second) parent. The second part beyond the cutting point is filled by copying the same sequence of the missing jobs in the other parent.

Mutation: Whether to mutate an individual or not is decided randomly according to a given mutation probability, μ. If an individual is selected for mutation, two jobs are selected randomly and their positions are interchanged. Mutation probability is determined by computational tests.

Termination criteria: GA is terminated after γ_g non-improving generations. This parameter is also determined by computational tests.

3.2. Robot Move Scheduling

When the part sequence is determined by the GA, the corresponding robot move cycle is determined and its cycle time is calculated by a TS algorithm. The details of the TS algorithm are as follows:

Initial solution: The initial robot move sequence is determined randomly. Starting with A_0, activities are sequenced randomly. However, at each iteration, the feasibility conditions presented in Section 2 is maintained. For an m-machine robotic cell in which

the MPS contains n parts, the final robot move sequence must contain $n(m+1)$ activities. Each of these activities is associated with a part in the MPS. Hence, in the solution encoding the activity sequence and the associated parts are represented together as shown in the following example with $m = 4$ and $n = 3$:

$$\sigma_{RMS} = \{A_0^{j_1}, A_3^{j_3}, A_1^{j_1}, A_0^{j_2}, A_2^{j_1}, A_1^{j_2}, A_4^{j_3}, A_3^{j_1}, A_0^{j_3}, A_4^{j_1}, A_2^{j_2}, A_1^{j_3}, A_3^{j_2}, A_4^{j_2}, A_2^{j_3}\}$$

In this example, the robot takes the first part of the MPS (j_1) from the input buffer and loads the to M_1. Then, moves to M_3 and unloads the last part of the MPS sequence from this machine and loads to M_4. Note that, we consider the steady state repetition of the cycles. In this example, at the beginning of the cycle, M_3 is loaded with j_3 and all other machines are empty. At the end of the cycle, three parts are dropped to the output buffer, M_3 is loaded with j_3, and all other machines are empty again. When the part sequence and the robot move sequence are given, the activities and the parts can easily be associated with each other by assuming (without loss of generality) that the first A_0 activity is always associated with j_1.

Neighborhood definition: At each iteration, the TS algorithm checks the neighbors of the current solution and moves to the best non-tabu neighbor. The performance of the algorithm depends highly on the neighborhood definition. In this study, we use the insertion operator to generate neighbors. However, since the generated neighbors must be feasible, the moves are constrained by the feasibility conditions. In order to generate all neighbors of a given activity sequence, each activity is considered one by one starting from the first activity in the sequence (A_0). Let the currently considered activity be A_j. The next repetition of the same activity is found and its position is determined. Let this position be p. Then, by moving A_j to positions $p-1, p-2, \ldots$ new neighboring activity sequences are determined and recorded. This is continued until reaching an A_{j-1} or an A_{j+1} activity. Since the feasibility conditions state that between any two consecutive repetitions of A_j there must be exactly one A_{j-1} and one A_{j+1} activity, the A_j activity cannot be moved in front of any one of these two activities. Similarly, A_j is also moved in the other direction to positions $p+1, p+2, \ldots$ until reaching any one of the A_{j-1} or A_{j+1} activities. If the end of the activity sequence is reached before finding any one of A_{j-1} or A_{j+1}, the search continues from the beginning of the activity sequence.

As an example consider the σ_{RMS} activity sequence given above and consider $A_1^{j_1}$ in position 3. The next repetition of A_1 is in position 6. Since there is an A_2 in position 5, $A_1^{j_2}$ cannot be moved forward. But it can be moved backward until position 8 where there is an A_0. Therefore, the neighbors that we can attain by considering $A_1^{j_1}$ are the following two activity sequences:

$$\sigma_{N1} = \{A_0^{j_1}, A_3^{j_3}, A_1^{j_1}, A_0^{j_2}, A_2^{j_1}, A_4^{j_3}, \mathbf{A_1^{j_2}}, A_3^{j_1}, A_0^{j_3}, A_4^{j_1}, A_2^{j_2}, A_1^{j_3}, A_3^{j_2}, A_4^{j_2}, A_2^{j_3}\}$$

$$\sigma_{N2} = \{A_0^{j_1}, A_3^{j_3}, A_1^{j_1}, A_0^{j_2}, A_2^{j_1}, A_4^{j_3}, A_3^{j_1}, \mathbf{A_1^{j_2}}, A_0^{j_3}, A_4^{j_1}, A_2^{j_2}, A_1^{j_3}, A_3^{j_2}, A_4^{j_2}, A_2^{j_3}\}$$

All activities are considered in a similar way and all possible neighbors are generated.

Calculation of the cycle time:

When the part sequence σ_{PS} and the robot move sequence σ_{RMS} are determined, the associated decision variables in MILP1 and/or MILP2 can be fixed. This converts both formulations into linear programs. The resulting cycle time can be calculated by solving any one of these LPs. However, here we develop an iterative procedure which calculates the cycle time without requiring to solve any mathematical program. In order to present the iterative algorithm, let the part sequence be given as $\{j_1, j_2, \ldots, j_n\}$ and let us denote the activity in position k by $A_{i_k}^{j_k}$, meaning that this activity unloads part j_k from machine i_k and moves it to machine $i_k + 1$. As a consequence, the robot move sequence and the associated parts can be represented as $\sigma_{RMS} = \{A_{i_1}^{j_1}, A_{i_2}^{j_2}, \ldots, A_{i_{n(m+1)}}^{j_{n(m+1)}}\}$. On the other hand, let the position of the activity that unloads part j from machine i is denoted by $\xi(i, j)$.

Let v be the iteration counter. The algorithm starts by initializing the counter ($v = 0$), all activity starting times, $T_k^v = 0$, $\forall k$, and the cycle time, $CT^v = 0$. Then it steps the counter up, sets $T_1^v = 0$, and recalculates the starting times and the resulting cycle time at each iteration using the following recursive equation:

$$T_k^v = \left\{ \begin{array}{ll} 0, & \text{if } k = 1, \\ 2\epsilon + \delta + T_{k-1}^v + |i_k - (i_{k-1} + 1)|\delta, & \text{if } i_k = 0, \\ 2\epsilon + \delta + \max\{T_{k-1}^v + |i_k - (i_{k-1} + 1)|\delta, T_{\xi(i_k-1,j_k)}^v + p_{i_k j_k}\}, & \\ & \text{if } i_k \neq 0 \text{ and } k > \xi(i_k - 1, j_k), \\ 2\epsilon + \delta + \max\{T_{k-1}^v + |i_k - (i_{k-1} + 1)|\delta, T_{\xi(i_k-1,j_k)}^{v-1} + p_{i_k j_k} - CT^{v-1}\}, & \\ & \text{if } i_k \neq 0 \text{ and } k < \xi(i_k - 1, j_k). \end{array} \right\} k = 1, 2, \ldots, n(m+1), \tag{26}$$

The cycle time is calculated at the end of iteration v as $CT^v = T^v_{n(m+1)} + 2\epsilon + \delta + (i_{n(m+1)}+1)\delta$. The recursive equation utilizes the fact that the starting time of the activity at position k (T_k) must be later than the starting time of the activity in position $k-1$ (T_{k-1}) plus the necessary time for the robot to complete that activity and reach machine i_k, $(2\epsilon+\delta+|i_k-(i_{k-1}+1)|\delta)$. This is the first part of the equation which is identical in all three cases. On the other hand, T_k must also be later than the starting time of the same part in the previous machine ($T_{\xi(i_k-1,j_k)}$) plus the necessary time to complete that activity and load machine i_k ($2\epsilon+\delta$) plus the processing time of the part on this machine ($p_{i_k j_k}$). However, if the activity at position k is A_0 ($i_k = 0$), then there is no previous machine. Hence, this second equation diminishes and the first case of the recursive equation arises. On the other hand, since the robot moves are cyclic, the unloading activity of a part on M_i can be in a later or in an earlier position than the unloading activity of the same part on M_{i-1}. The next two cases of the recursive equation consider these situations, respectively. The latter one includes CT to correctly represent the relation between the starting times.

This procedure is stopped as soon as the cell converges to a steady state in which the starting times of the corresponding activities become identical in different repetitions of the cycle. That is, the procedure is stopped at the first iteration v^* in which $\exists u = 1, 2 \ldots v^* - 1$ such that $T^{v^*}_k = T^{v^*-u}_k, \forall k$. Hall et al. (1997) considered CRM cycles separately in a 3-machine robotic cell and showed that the cell converges to a steady state in a number of MPS cycles starting from the empty state in which all machines are empty. Furthermore, they proved that for some CRM cycles, a single MPS is not satisfactory to define a steady state, but two MPS's may be required for this purpose. Let u^* be the minimum u value that satisfies the above equality. Therefore, u^* is the number of MPS's that is required to define a steady state. Our computational tests revealed that the cell converges to a steady state after very few iterations and at most two MPS's are required to define a steady state. After the procedure is stopped the resulting cycle time is calculated as the average cycle time of multiple MPS's that are required to define a steady state, $CT = \sum_{l=0}^{u^*-1} CT^{v^*-l}/u^*$.

The following example demonstrates this calculation procedure:

Example 2. Let us consider a 2-machine robotic cell in which the MPS contains 3 parts. Let $\epsilon = 1$, $\delta = 2$, $p_{11} = 3$, $p_{21} = 6$, $p_{12} = 5$, $p_{22} = 2$, $p_{13} = 7$, and $p_{23} = 4$. Let us calculate the cycle time for $\sigma_{PS} = \{1, 2, 3\}$ and $\sigma_{RMS} = \{A_0^1, A_2^3, A_1^1, A_0^2, A_2^1, A_1^2, A_2^2, A_0^3, A_1^3\}$. According to this the recursive equations can be written as follows:

	$v=0$	$v=1$	$v=2$
$T_1^v = 0$	0	0	0
$T_2^v = 2\epsilon + \delta + \max\{T_1^v + \|2-(0+1)\|\delta, T_9^{v-1} + p_{23}\}$	0	6	6
$T_3^v = 2\epsilon + \delta + \max\{T_2^v + \|1-(2+1)\|\delta, T_1^v + p_{11}\}$	0	14	14
$T_4^v = 2\epsilon + \delta + T_3^v + \|0-(1+1)\|\delta$	0	22	22
$T_5^v = 2\epsilon + \delta + \max\{T_4^v + \|2-(0+1)\|\delta, T_3^v + p_{21}\}$	0	28	28
$T_6^v = 2\epsilon + \delta + \max\{T_5^v + \|1-(2+1)\|\delta, T_4^v + p_{12}\}$	0	36	36
$T_7^v = 2\epsilon + \delta + \max\{T_6^v + \|2-(1+1)\|\delta, T_6^v + p_{22}\}$	0	42	42
$T_8^v = 2\epsilon + \delta + T_7^v + \|0-(2+1)\|\delta$	0	52	52
$T_9^v = 2\epsilon + \delta + \max\{T_8^v + \|1-(0+1)\|\delta, T_8^v + p_{13}\}$	0	63	63

Since $T_k^1 = T_k^2$, $\forall k$, the procedure stops with $v^* = 2$ and $u^* = 1$. The resulting cycle time is $CT = T_9^2 + 2\epsilon + \delta + |1+1|\delta = 71$.

Tabu list: Length of the tabu list is one of the most important parameters of TS used to balance intensification and diversification of the search. When the search moves to a neighboring solution, that move is forbidden for several iterations in order to avoid cycling. The length of the tabu list is selected as $\lambda.n.(m+1)$, where $0 \leq \lambda \leq 1$ is a constant. This formula adjusts the length of the tabu list according to the problem size. λ is determined by computational tests.

Termination condition : TS is terminated after γ_t non-improving iterations. This parameter is also determined by computational tests.

3.3. Hybrid Metaheuristic for CRM Cycles

The flowchart of the heuristic given in Figure 4 is identical for the CRM case. All the mechanisms of the GA and the TS are exactly the same for the CRM case except

the procedure for finding the initial solution and the neighborhood definition of the TS. As already defined, a CRM cycle is the repetition of the same 1-unit cycle n times to produce all the parts in the MPS. Consider a four-machine robotic cell producing three different parts. The following robot move sequence is a CRM cycle example that repeats the 1-unit cycle $\{A_0, A_3, A_4, A_2, A_1\}$ three times with the part sequence (j_1, j_2, j_3):

$$\sigma_{RMS} = \{A_0^{j_1}, A_3^{j_2}, A_4^{j_2}, A_2^{j_3}, A_1^{j_1}, A_0^{j_2}, A_3^{j_3}, A_4^{j_3}, A_2^{j_1}, A_1^{j_2}, A_0^{j_3}, A_3^{j_1}, A_4^{j_1}, A_2^{j_2}, A_1^{j_3}\}$$

Starting with the A_0 activity, any permutation of the activities A_i, $i = 1, 2, \ldots, m$ leads to a feasible 1-unit cycle (Crama and van de Klundert, 1997). Hence, there is no need to check the feasibility conditions. As a consequence, the initial solution is found by randomly ordering the activities to generate a 1-unit cycle. Then, the same activity sequence is concatenated n times. Similarly, in order to determine the neighbors of a given solution, the base 1-unit cycle is considered. The insertion move is similar to the MPS case in which each activity is removed from its current position and placed to a different position. There is no need to check the feasibility. The resulting 1-unit cycle is again concatenated n times and the cycle time is calculated using the same iterative procedure as explained above.

The parameters used in the GA and the TS are determined separately for the CRM and MPS cases with a computational study as explained in the following section.

3.4. Parameter Calibration

In this section, we perform computational tests to determine the best values for all the parameters of the hybrid algorithm. The parameter calibration is performed for the MPS and the CRM cases separately. Here we explain the MPS case in detail. The same procedure is repeated for the CRM case also. The heuristic is coded in Microsoft Visual Studio C++ on a machine with Intel Xeon E5645 2.40 GHz CPU (2×6 cores), 18 GB of RAM under a 64bit operating system.

For parameter calibration two different test problems are used. The first one is taken from Carlier (1978) that includes the processing times for $m = n = 7$ flow shop problem. The second problem is an $m = 5$, $n = 10$ problem in which the processing times are randomly generated uniformly between $(0, 10000)$. Besides the number of machines and

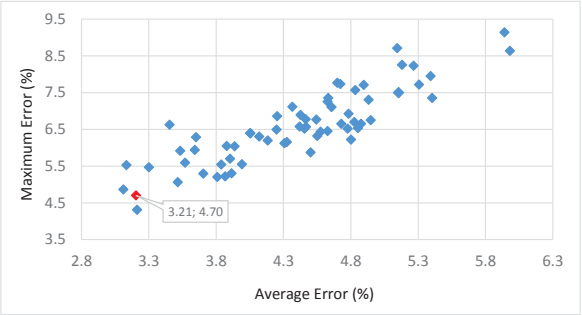
the parts, the main difference between these two test problems is the variance of the processing times. For each test problem $\epsilon = 1$ is used and two different δ values are selected with respect to the processing times on the machines. As a result 20 and 50 are used for 7×7 problems and 30 and 75 are used for 5×11 problems. As a consequence, there are four test problems. In order to decide on the best parameter values, two levels are used for each parameter as listed in Table 5. Each combination is solved for five times in order to minimize the effect of randomness included in the heuristic. As a result, $4 \times 2^6 \times 5 = 1280$ problems are solved.

Table 5: The levels of parameters

Parameters	**Levels**	
β	10	15
θ	0.1	0.2
μ	0.05	0.1
γ_g	50	75
λ	0.05	0.1
γ_t	30	50

The performance of each parameter combination is measured in terms of the average and maximum % errors and the CPU time. % error of a replication is calculated by comparing its cycle time with the best cycle time attained for the corresponding problem with any parameter combination. Then, the average and the maximum of the five replications are calculated. The attained results for each of the 64 parameter combinations are depicted in Figure 5. Figure 5a depicts the maximum errors with respect to the average errors and Figure 5b depicts the CPU times with respect to the average errors. We selected the parameter combination that leads to an average error of 3.21% and a maximum error of 4.70% which are marked in Figures 5a and 5b. This alternative is selected since it is close to the best in terms of the average and the maximum errors but has a better solution time (56.18s) than the best possible time. The selected parameter values are $\beta = 15$, $\theta = 0.2$, $\mu = 0.05$, $\gamma_g = 75$, $\lambda = 0.05$, and $\gamma_t = 50$.

When the same procedure is repeated for the CRM case, the best parameter combination is appeared to be $\beta = 10$, $\theta = 0.1$, $\mu = 0.1$, $\gamma_g = 75$, $\lambda = 0.05$, and $\gamma_t = 50$. The resulting average error is 1.41% and the maximum error is 2.28% when compared with



(a) Average vs. maximum percent error

(b) CPU time vs. average percent error

Figure 5: Parameter calibration results for the MPS cycles

the best cycle time attained among the CRM cycles. When compared with the best MPS cycle, the average and maximum errors are 7.46% and 8.35%, respectively. The CPU time is 4.34s.

4. Computational Study

The objectives of the computational study are threefold. First, to evaluate the performance of the hybrid metaheuristic algorithm by comparing it with the MILP2 solutions. Second, to assess the benefits of considering MPS cycles instead of CRM cycles. Finally, to assess the benefits of parallelization of the hybrid metaheuristic algorithm. We denote the parallel version of the heuristic for MPS (CRM) cycles with MPS-P (CRM-P), and the single thread version with MPS-S. All algorithms are coded in C++ and compiled with Microsoft Visual Studio on a machine with Intel Xeon E5645 2.40 GHz CPU (2×6 cores),18 GB of RAM under a 64bit operating system. As already mentioned, the mathematical models are solved with GAMS CPLEX 12.6.2 solver with a one-hour time limit.

There are no test problems available in the literature for this problem. Therefore we used two methods to generate the test problems. First, since we consider a flow shop type system, we used available test problems for flow shop scheduling problems. These test problems include the processing times of the parts on the machines. We attached robot travel times (δ) and machine loading/unloading times (ϵ) to these problems. We used the eight test problems from Carlier (1978) for this method. Since the total number

of loading/unloading operations are identical for each cycle, the magnitude of ϵ does not affect the problem complexity. Therefore, we assumed $\epsilon = 1$ in all problem instances. On the other hand, since the number of travel moves is different in each cycle, δ has an effect on the problem complexity. We generated two different δ values for each of the eight problems. These values are generated such that when δ is small, the second term of the max function determines the LB in Equation (20) and when δ is large, the first term determines the LB. 16 test problems are generated by this method.

As a second method, we considered all combination of machines and parts for $n \in \{3, 4, 5, 6\}$ and $m \in \{3, 4, 5\}$ and generated all processing times randomly. Since our initial tests revealed that the problem complexity increases with the variance of the processing times, we generated the processing times uniformly from the interval $[1, 1000]$ for these 12 problems. For each of these instances, we then generated two different δ values as explained above that counts to 24 problems. Together with the former 16 problems, we have a total of 40 test problems.

All test problems are solved with MPS-P, CRM-P, and MPS-S algorithms and the MILP2 model. Each problem instance is solved five times with these algorithms in order to reduce the effect of randomness. The results are summarized in Table 6. MILP2 was not able to determine the optimal solution in 30 of the test problems in one-hour time limit. In 22 of these, it was unable to find even an integer feasible solution. In order to be able to evaluate the performance of the heuristics for these instances, we utilized the best solution generated by MPS-P, CRM-P, and MPS-S into the MILP2 model as an initial solution and ran it for an additional one-hour time limit. By this method, we were able to generate the optimal solutions for two more instances and we could calculate the relative percent gap in the remaining 22 instances. We could improve the gap value in a total of 26 instances. Improved gap values are labeled with a "$*$" in Table 6.

We compared the heuristic results with the optimal or the best integer values provided by MILP2. Since five replications are taken with the heuristics we calculated two different percent error terms, E1 and E2 as follows:

$$\text{E1} = 100 \times (\text{Average of 5 replications} - \text{Best Integer})/\text{Best Integer}$$

$$\text{E2} = 100 \times (\text{Minimum of 5 replications} - \text{Best Integer})/\text{Best Integer}$$

Table 6: Summary of the results

	MPS-P			MPS-S	CRM-P			MILP2	
(n, m, δ)	E1 (%)	E2 (%)	CPU (s)	CPU (s)	E1 (%)	E2 (%)	CPU (s)	Gap (%)	CPU (s)
(3,3,50)	0.0	0.0	0.4	1.0	0.0	0.0	0.2	0.0	1.9
(3,3,100)	0.0	0.0	0.5	1.4	1.9	1.9	0.2	0.0	1.7
(4,3,50)	0.0	0.0	0.7	2.1	4.3	4.3	0.3	0.0	31.2
(4,3,100)	0.0	0.0	0.9	2.7	2.3	2.3	0.3	0.0	22.7
(5,3,30)	0.2	0.0	1.2	3.9	3.3	3.3	0.4	0.0	536.9
(5,3,75)	0.5	0.0	1.3	4.1	11.3	11.3	0.4	0.0	481.8
(6,3,50)	2.3	0.0	1.6	5.4	4.3	4.2	0.4	8.7	TL
(6,3,100)	0.0	0.0	1.6	5.6	3.7	3.7	0.4	7.0	TL
(3,4,30)	1.2	0.0	0.8	2.1	3.4	1.5	0.4	0.0	58.5
(3,4,60)	0.0	0.0	1.0	2.7	0.3	0.3	0.4	0.0	65.8
(4,4,30)	0.6	0.0	1.2	4.8	10.0	2.8	0.6	0.0*	TL
(4,4,60)	0.0	0.0	1.5	4.8	11.9	11.9	0.5	9.6	TL
(5,4,50)	0.0	0.0	1.8	7.0	10.1	0.0	0.7	0.0*	TL
(5,4,100)	0.4	0.0	3.1	10.5	5.7	5.7	0.8	11.6	TL
(6,4,40)	1.3	0.0	2.8	14.9	11.0	5.8	1.0	26.7*	TL
(6,4,80)	1.5	0.0	3.5	16.8	7.7	7.7	0.8	8.2*	TL
(3,5,30)	5.2	3.1	1.2	3.9	15.1	5.7	0.9	0.0	1900
(3,5,60)	2.0	0.0	1.2	4.4	10.7	5.4	0.7	0.0	1500
(4,5,40)	2.5	0.0	1.9	9.6	12.0	11.3	1.1	17.8*	TL
(4,5,80)	0.6	0.0	2.8	13.5	7.6	7.6	0.9	11.0*	TL
(5,5,40)	4.2	0.0	4.0	22.7	16.8	6.6	1.3	18.0*	TL
(5,5,75)	1.1	0.0	5.1	21.6	11.8	11.4	1.4	8.3*	TL
(6,5,40)	1.9	0.0	9.0	46.0	14.3	14	1.4	6.0*	TL
(6,5,75)	0.6	0.0	8.7	43.2	4.7	4.7	1.4	7.3*	TL
(11,5,30)	2.6	0.0	48.5	271.9	17.7	8.3	6.2	13.7*	TL
(11,5,75)	1.7	0.0	80.3	373.0	12.7	10.2	2.9	8.0*	TL
(13,4,30)	3.0	0.0	38.1	272.2	15.6	7.2	2.3	12.1*	TL
(13,4,100)	1.0	0.0	49.1	355.7	8.5	7.8	2.4	7.8*	TL
(12,5,25)	2.0	0.0	42.4	381.9	15.8	6.2	4.6	24.0*	TL
(12,5,75)	1.0	0.0	70.1	461.3	13.2	7.9	4.0	10.7*	TL
(14,4,30)	2.0	0.0	63.7	298.7	15.7	8.4	3.1	17.7*	TL
(14,4,75)	0.9	0.0	50.9	345.8	10.3	6.4	2.7	9.4*	TL
(10,6,25)	1.7	0.0	101.3	468.2	21.2	17	4.5	23.2*	TL
(10,6,60)	2.0	0.0	65.2	424.1	14.7	13.5	6.1	10.5*	TL
(8,9,20)	3.8	0.0	147.9	690.7	10.9	5.3	24.7	40.6*	TL
(8,9,50)	2.0	0.0	127.5	863.3	6.5	6.1	24.5	14.9*	TL
(7,7,20)	1.6	0.0	29.5	161.1	17.3	8.1	5.6	34.1*	TL
(7,7,50)	0.7	0.0	48.0	172.6	9.9	4.4	8.4	18.0*	TL
(8,8,20)	5.1	0.0	90.6	513.4	19.1	16.8	13.0	26.2*	TL
(8,8,50)	0.5	0.0	124.4	477.8	9.0	8.0	11.6	16.6*	TL
Avg	1.4	0.1			10.1	6.9			
Max	5.2	3.1			21.2	17			

* : Improved gap values TL: Time Limit

Since, MPS-P and MPS-S are the same algorithms running in parallel and single thread modes, only the solution times for the MPS-S are reported in Table 6. As it can

be seen in this table, MPS-P provides high-quality solutions with average and maximum E1 values of 1.4% and 5.2%. The average error is higher when the δ values are smaller (2.1% vs. 0.8%). In all instances except one, the best of MPS-P is equal to the best integer provided by MILP2.

In 30 of the instances, MILP2 hit the one-hour time limit. The average gap of MILP2 is 13.46% when δ is small and 7.94% when it is large. On the other hand, the CPU time of MPS-P is less than 10 seconds when $n \leq 6$ and $m \leq 5$ and it is reasonable in all other instances for such a complex problem. When MPS-P and MPS-S are compared with each other, we can conclude that parallelization has a great impact on the solution time. Our tests revealed that the solution time can be reduced to one-quarter of its original value by parallelization. This result depends on the number of CPU cores of the computer.

It is observed that the cycle times of the MPS cycles are approximately 9% smaller than the corresponding CRM cycles on the average. The maximum benefit is close to 20%. The benefits are greater when the δ values are smaller (9.6% vs. 7.3%). On the other hand, the average CPU time of MPS-P is approximately 7 times larger than CRM-P.

5. Conclusion

In this study, we considered the scheduling of the parts and robot moves in an m-machine robotic cell. We developed a mixed integer linear programming formulation that determines the optimal part sequence and the MPS cycle simultaneously. By utilizing several valid inequalities and reformulations, we improved this formulation. However, since the problem complexity increases with respect to the number of machines and the number of parts in the MPS, we developed a hybrid metaheuristic algorithm that combines genetic algorithms with tabu search. The algorithm is modified in order to be able to solve the problems with the CRM cycles as well. In order to improve the performance of the heuristic we utilized thread parallelization. The parameters of the heuristic is calibrated through experimental tests. An extensive computational study revealed that the parallel hybrid metaheuristic provides high quality solutions in reasonable CPU times. It is also shown that parallelization has a significant impact on solution time of the heuristic. In our case, it reduced the solution time to one quarter of its original value. Another important output of this study is the quantification of the benefits of considering MPS cycles instead

of the CRM cycles. The CRM cycles are simple, practical, and easy to control and implement. However, our results show that the MPS cycles can increase the throughput rate up to 20% depending on the problem parameters. The improvement is approximately 9% on the average.

This study can be extended by considering robots with different technological properties and capabilities such as dual arm robots (Geismar et al., 2012) or robots with a self buffer (Gundogdu and Gultekin, 2016). However, in both cases the number of feasible robot move cycles increases drastically which increases the problem complexity even further. Developing MILP formulations and heuristic algorithms will contribute to the literature. The problem can also be handled with other widely used scheduling objectives such as total completion time and tardiness. The problem can also be considered with multiple objectives.

Acknowledgements

This research is supported by the Scientific and Technological Research Council of Turkey under grant #213M435.

References

M. Abdulkader, M. ElBeheiry, N. Afia, and A. El-Kharbotly. Scheduling and sequencing in four machines robotic cell: Application of genetic algorithm and enumeration techniques. *Ain Shams Engineering Journal*, 4(3):465 – 474, 2013.

Y. P. Aneja and H. Kamoun. Scheduling of parts and robot activities in a two machine robotic cell. *Computers & Operations Research*, 26(4):297–312, 1999.

G. D. Batur, O. E. Karasan, and M. S. Akturk. Multiple part-type scheduling in flexible robotic cells. *International Journal of Production Economics*, 135(2):726–740, 2012.

N. Brauner and G. Finke. On cycles and permutations in robotic cells. *Mathematical and Computer Modelling*, 34:565–591, 2001.

J. Carlier. Ordonnancements à contraintes disjonctives. *RAIRO - Operations Research - Recherche Opérationnelle*, 12(4):333–350, 1978.

J. Carlier, M. Haouari, M. Kharbeche, and A. Moukrim. An optimization-based heuristic for the robotic cell problem. *European Journal of Operational Research*, 202(3):636–645, 2010.

A. Che, P. Yan, N. Yang, and C. Chu. Optimal cyclic scheduling of a hoist and multi-type parts with fixed processing times. *International Journal of Production Research*, 48(5): 1225–1243, 2010.

A. Che, Z. Zhou, C. Chu, and H. Chen. Multi-degree cyclic hoist scheduling with time window constraints. *International Journal of Production Research*, 49(19):5679–5693, 2011.

Y. Crama and J. van de Klundert. Cyclic Scheduling of Identical Parts in a Robotic Cell. *Operations Research*, 45(6):952–965, 1997.

M. Dawande, H. Geismar, S. Sethi, and C. Sriskandarajah. *Throughput Optimization in Robotic Cells*. International Series in Operations Research & Management Science. Springer US, 2007.

A. Elmi and S. Topaloglu. Scheduling multiple parts in hybrid flow shop robotic cells served by a single robot. *International Journal of Computer Integrated Manufacturing*, 27(12):1144–1159, 2014.

A. Elmi and S. Topaloglu. Multi-degree cyclic flow shop robotic cell scheduling problem: Ant colony optimization. *Computers and Operations Research*, 73:67–83, 2016.

M. Fazel Zarandi, H. Mosadegh, and M. Fattahi. Two-machine robotic cell scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, 40 (5):1420–1434, 2013.

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

N. Geismar, U. V. Manoj, A. Sethi, and C. Sriskandarajah. Scheduling robotic cells served by a dual-arm robot. *IIE Transactions*, 44(3):230–248, 2012.

H. Gultekin, M. S. Akturk, and O. E. Karasan. Scheduling in robotic cells: process flexibility and cell layout. *International Journal of Production Research*, 46(8):2105–2121, 2008.

H. Gultekin, O. E. Karasan, and M. S. Akturk. Pure cycles in flexible robotic cells. *Computers & Operations Research*, 36(2):329 – 343, 2009.

E. Gundogdu and H. Gultekin. Scheduling in two-machine robotic cells with a self-buffered robot. *IIE Transactions*, 48(2):170–191, 2016.

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

N. G. Hall, H. Kamoun, and C. Sriskandarajah. Scheduling in robotic cells: Complexity and steady state analysis. *European Journal of Operational Research*, 109(1):43 – 65, 1998.

I. Kamalabadi, S. Gholami, and A. Mirzaei. A new solution for the cyclic multiple-part type three-machine robotic cell problem based on the particle swarm meta-heuristic. *Journal of Industrial and Systems Engineering*, 1(4):304–317, 2008.

V. Kats, E. Levner, and L. Meyzin. Multiple-part cyclic hoist scheduling using a sieve method. *IEEE Transactions on Robotics and Automation*, 15(4):704–713, 1999.

X. Li and R. Y. K. Fung. A mixed integer linear programming solution for single hoist multi-degree cyclic scheduling with reentrance. *Engineering Optimization*, 46(5):704–723, 2014.

S. Q. Liu and E. Kozan. A hybrid metaheuristic algorithm to optimise a real-world robotic cell. *Computers & Operations Research*, 2016. doi: http://dx.doi.org/10.1016/j.cor.2016.09.011.

M. Nawaz, E. Enscore, and I. Ham. A heuristic algorithm for the m-machine n-job flow-shop sequencing problem. *OMEGA*, 11(1):91–95, 1983.

Robotics Online. *North American Robotics Market Continues Breaking Records with Strongest Opening Quarter Ever*, 2016. URL `http://www.robotics.org/content-detail.cfm/Industrial-Robotics-News/North-American-Robotics-Market-Continues-Breaking-Records-with-Strongest-Opening-Quarter-Ever/content_id/5367`. Accessed: 05.11.2016.

R. Ruiz and C. Maroto. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research*, 169(3): 781 – 800, 2006.

S. P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak. Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems*, 4(3-4):331–358, 1992.

A. Soukhal and P. Martineau. Resolution of a scheduling problem in a flowshop robotic cell. *European Journal of Operational Research*, 161(1):62–72, 2005.

C. Sriskandarajah, N. G. Hall, and H. Kamoun. Scheduling large robotic cells without buffers. *Annals of Operations Research*, 76(0):287–321, 1998.

C. Sriskandarajah, I. Drobouchevitch, S. P. Sethi, and R. Chandrasekaran. Scheduling multiple parts in a robotic cell served by a dual-gripper robot. *Operations Research*, 52 (1):65–82, 2004.

W. Zahrouni and H. Kamoun. Sequencing and scheduling in a three-machine robotic cell. *International Journal of Production Research*, 50(10):2823–2835, 2012.

Z. Zhou, A. Che, and P. Yan. A mixed integer programming approach for multi-cyclic robotic flowshop scheduling with time window constraints. *Applied Mathematical Modelling*, 36(8):3621–3629, 2012.