Available online at www.sciencedirect.com

ScienceDirect

Computers & Operations Research 36 (2009) 329 – 343

computers & operations research

www.elsevier.com/locate/cor

Pure cycles in flexible robotic cells

Hakan Gultekin, Oya Ekin Karasan, M. Selim Akturk*

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

Available online 10 October 2007

Abstract

In this study, an m-machine flexible robotic manufacturing cell consisting of CNC machines is considered. The flexibility of the machines leads to a new class of robot move cycles called the pure cycles. We first model the problem of determining the best pure cycle in an m-machine cell as a special travelling salesman problem in which the distance matrix consists of decision variables as well as parameters. We focus on two specific cycles among the huge class of pure cycles. We prove that, in most of the regions, either one of these two cycles is optimal. For the remaining regions we derive worst case performances of these cycles. We also prove that the set of pure cycles dominates the flowshop-type robot move cycles considered in the literature. As a design problem, we consider the number of machines in a cell as a decision variable. We determine the optimal number of machines that minimizes the cycle time for given cell parameters such as the processing times, robot travel times and the loading/unloading times of the machines.
© 2007 Elsevier Ltd. All rights reserved.

Keywords: Flexible manufacturing systems; CNC; Industrial robots; Cellular automation; Optimization; Production control

1. Introduction

A manufacturing cell which consists of a number of machines and a material handling robot is called a robotic cell. An m-machine robotic cell can be seen in Fig. 1. Such manufacturing cells are used extensively in chemical, electronic and metal cutting industries. In this study, we will restrict ourselves with the metal cutting applications in an environment in which the machines are predominantly CNC machines so that the machines and the robot can communicate in a real-time basis. These machines are highly flexible and capable of performing several different operations by fast and inexpensive tool changes as long as the required tools are loaded in their tool magazines. There are no buffers at or between the machines. Hence, at any time instant, a part is either on one of the machines, on the robot or at the input or output buffer. Each of the identical parts to be produced is assumed to have a number of operations to be performed on the machines. As a consequence of the flexibility of the machines, these operations can be performed in any order on each of the machines. Furthermore, each operation can be assigned to any one of the machines. In order to use such systems efficiently, problems including the scheduling of the robot moves and the determination of the machines to perform each operation of each part should be solved. Throughout this study, these problems will be tackled with the objective of maximizing the throughput rate.

There is an extensive literature on robotic cell scheduling problems as summarized in the surveys of Crama et al. [1] and Dawande et al. [2]. Most of the research on this area assumed the cell to work as a flowshop-type system. More formally, each part is assumed to visit all of the machines in the same order, machine 1 through machine m in

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

0305-0548/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2007.10.007

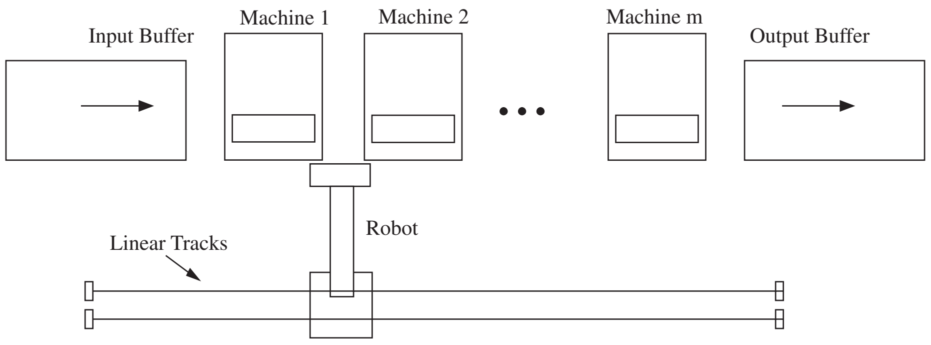

Fig. 1. m-Machine robotic cell.

an m-machine cell. Although this assumption might be valid for chemical or electroplating operations, it unnecessarily limits the number of alternative solutions in a flexible manufacturing cell (FMC), such as the one studied in this paper. Sethi et al. [3] developed the necessary framework for these scheduling problems and proved that for two machine producing identical parts, the optimal solution is a 1-unit cycle, where an n-unit cycle is defined to be a robot move cycle in which, starting with an initial state, the robot loads and unloads all of the machines exactly n times and returns back to the initial state. Note that in an n-unit cycle exactly n parts are produced. A similar result for three-machine case was proved by Crama and Van de Klundert [4]. However, the optimal solution is not necessarily a 1-unit cycle when the number of machines is greater than three [5]. Flexible robotic cells have recently been a topic of research. For example, in Akturk et al. [6], a robotic cell with two identical CNC machines possessing operational and process flexibility was considered. Operational flexibility is defined as the ability to interchange the ordering of several operations and process flexibility is defined as the ability to perform multiple operations on the same machine. For this problem, they proved that the optimal solution is either one of the two 1-unit cycles or the only 2-unit cycle.

In this study, we consider a new class of robot move cycles, named the pure cycles, resulting from the flexibility of the machines. Pure cycles are defined as the robot move cycles in which the robot loads and unloads all m machines with a different part during one repetition of the cycle. The terminology “pure” is to reflect the fact that each part is completely performed by only one machine and no part is transferred from one machine to another. Part movement is from the input buffer to one of the m machines and from this machine to the output buffer. A different sequence of loading and unloading operations leads to a different pure cycle. In earlier studies, we defined these cycles and showed that they perform efficiently in two-machine [7] and three-machine cells [8] in comparison to flowshop-type robot move cycles. These results are achieved by comparing one of the most simple and practical cycles among the class of pure cycles with the flowshop-type robot move cycles. However, the general problem of determination of the best pure cycle in an m-machine robotic cell was not tackled before. In this study we consider this problem.

This problem is somehow related with the parallel machine scheduling problem with a common server which can be reviewed in Hall et al. [9]. However, in that literature the setup time of the machines is arbitrary for each job and is given as a problem parameter. On the contrary, in our study, the setup time (transporting the part to the machine from the input buffer and loading it) is a variable depending on the robot move sequence. Additionally, different from that literature, the robot also performs the unloading of the machines. Finally, in that literature, it is assumed that a finite number of parts is to be produced and typically the objective function is either the minimization of the makespan or the total completion time. However, since we assume identical parts to be produced indefinitely and since the robot repeatedly follows a computer program, we consider cyclic scheduling. In a related study from this literature, Abdekhodaee et al. [10] considered scheduling of n different jobs on two parallel machines with the objective of minimizing the makespan where each job has its own processing and setup times given as problem parameters. Knowing that the general problem is NP-hard in the strong sense, they considered special cases of equal processing times and equal setup times. For the case of equal processing times, which is more related to our study, the authors prove that the problem is NP-hard in the ordinary sense when the setup times are small in comparison to the processing times and trivially solvable otherwise.

CNC machines possess several types of flexibilities. Such flexibilities are achieved by considering alternative tool types for operations and loading multiple tools to the tool magazines of the machines. This study focuses on the consequences of introducing such machine flexibilities to our system. We show that two specific pure cycles among the huge class of feasible robot move cycles perform significantly better than the others and derive the regions of optimality for these two cycles. For the remaining small region, we derive the worst case performance of these cycles. Furthermore, our study also provides a very useful managerial insight into the FMC design problem. We need to study the impact of design decisions, such as the number of identical CNC machines and the tool magazine capacity of each CNC machine, on system capabilities, since these critical design issues affect the productivity as well as the investment cost of the FMCs. We determine the optimal number of machines that minimizes the cycle time for both of these studied pure cycles. The machining of a typical part can require a sequence of operations using many tools. The finite capacity of tool magazines limits the set of operations that can be assigned to a machine. One of the possible disadvantages of the pure cycles in comparison with the classical flowshop-type robot move cycles is the fact that we allocate a copy of each required cutting tool to every CNC machine, which will increase the tool inventories. Furthermore, all the required tools to manufacture a certain part must be loaded to the tool magazine prior to the actual machining, which might necessitate a larger tool magazine (equivalently higher machine investment cost). As already discussed in Gray et al. [11], tool magazine capacity is among the most significant parameters for the determination of expected system throughput, yet little work has been done to evaluate the relative cost imposed on the system by the size of the tool magazine. Therefore, we compare the pure cycles with the classical flowshop-type robot move cycles to assess the marginal value of increasing tool magazine capacities on the cycle time and show that the proposed cycles dominate all flowshop-type robot move cycles for an *m*-machine robotic cell.

In the following section, the notation and basic assumptions pertinent to this study will be introduced. In Section 3, the pure cycles will be defined. Two specific pure robot move cycles will be distinguished and compared with the rest of such cycles and with the classical flowshop-type robot move cycles considered in the existing robotic cell scheduling literature in Section 4. Section 5 is devoted to the concluding remarks and future research directions.

2. Notation and assumptions

In this section we present the basic assumptions and the notation to be used throughout this study. Crama and Van de Klundert [12] introduce the following definition for the representation of the flowshop-type robot move cycles.

Definition 1. A_i is the robot activity defined as: robot unloads machine i, transfers part from machine i to machine $i+1$, loads machine $i+1$. The input buffer is denoted as machine 0 and the output buffer is denoted as machine $(m+1)$.

We shall proceed with an example in order to explain flowshop-type robot move cycles and how they are represented using these activities. Let us consider two-machine robotic cells for this example. There are two 1-unit cycles in a two-machine robotic cell: $S1$ cycle is represented by $A_0A_1A_2$ activity sequence and $S2$ cycle is represented by $A_0A_2A_1$ activity sequence. $S2$ cycle starts with the state where the first machine is waiting for a part to be loaded, the second machine is loaded with a part and the robot is in front of the input buffer just starting to take a part to load the first machine. The first activity is A_0; the robot takes a part from the input buffer and loads the first machine. Then in order to perform A_2, the robot travels from the first machine to the second machine. If the processing of the part is completed when the robot arrives in front of the machine it immediately unloads the part, otherwise waits in front of the machine to finish the processing and then unloads the part. Finally, it transports the part to the output buffer and drops the part. In order to perform A_1, it travels to the first machine and unloads the machine after waiting in front of it if necessary. Later, it transports the part to the second machine and loads it. In order to perform the A_0 activity of the next repetition of the cycle, the robot travels back to the input buffer. Just after reaching this machine, the initial and the final states become the same. Hence, the cycle is completed with one part produced. As is apparent, all parts pass through all of the machines in the same sequence. This is why these cycles are called flowshop-type robot move cycles.

Such a definition of robot activities is necessary and enough for the traditional research in this area where the system is assumed to be a flowshop. However, in this study, we assume that each machine has the capability of performing all of the operations of a part. This flexibility allows the possibility of new cycles which can be represented by modifying the robot activity defined above as follows:

Definition 2. L_i is the robot activity in which the robot takes a part from the input buffer and loads machine i, $i=1,2,\ldots,m$. Similarly, U_i , $i=1,2,\ldots,m$, is the robot activity in which the robot unloads machine i and drops the part to the output buffer. Let $\mathscr{A}=\{L_1,\ldots,L_m,U_1,\ldots,U_m\}$ be the set of all activities.

In an m-machine robotic cell there are exactly m loading and m unloading activities. We can define new cycles by using these activities as follows:

Definition 3. Under a *pure cycle*, starting with an initial state, the robot performs each of the $2m$ activities ($L_i, U_i, i = 1, \dots, m$) exactly once and the final state of the system is identical to the initial state.

Note that under these cycles all of the operations of each part are performed completely by one of the machines and between two loadings of any one of the machines, all other machines are loaded exactly once. Each permutation of the $2m$ activities defines a pure cycle. However, some permutations define the same pure cycle. For example, in two-machine case, $L_1L_2U_1U_2$ and $U_1U_2L_1L_2$ are different representations of the same cycle. In this study, without loss of generality we will assume that all cycles start with activity L_1. As a result, after eliminating the different representations, there exists a total of $(2m-1)!$ different pure cycles in an m-machine cell. This makes six cycles for two-machine cells and 120 cycles for three-machine cells. In order to clarify the definitions, we list below all possible cycles in a two-machine cell:

$L_1L_2U_1U_2$	$L_1L_2U_2U_1$
$L_1U_1L_2U_2$	$L_1U_1U_2L_2$
$L_1U_2U_1L_2$	$L_1U_2L_2U_1$

Let us analyse $L_1U_2L_2U_1$ cycle further in detail. Initially, the first machine is empty, the second machine is loaded with a part and the robot is in front of the input buffer just starting to take a part. The robot transports the part to the first machine and loads it. In order to perform U_2, it travels from the first machine to the second machine, unloads the second machine after waiting in front of it if necessary, transports the part to the output buffer and drops it. In order to perform L_2, the robot travels back to the input buffer, takes another part, transports it to the second machine and loads it. In order to perform U_1, it travels back to the first machine. If necessary, it waits in front of the machine, unloads it and drops the part to the output buffer. In order to start the next repetition of the cycle, the robot travels back to the input buffer. Just after reaching the input buffer, the cycle is completed. Note that two parts are produced in one repetition of this cycle (the animated views of some of the pure cycles and the flowshop-type robot move cycles can be found at the web site http://www.ie.bilkent.edu.tr/∼robot).

Before we proceed let us list the remaining notation to be used throughout the text:

C_i^m	ith pure cycle in an m-machine robotic cell.
P	total processing time of the parts on the machines.
ε	the load and unload time 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, travelling from machine i to machine j is equal to $\lvert i-j\rvert\delta$.
$T_{C_i^m}$	cycle time of cycle C_i^m, that is, the total time required to complete the cycle.
$\Omega_{C_i^m}$	per unit cycle time of the cycle C_i^m, that is, the long run average time to produce one part.

One repetition of a pure cycle produces m parts. Hence, $\Omega_{C_i^m} = T_{C_i^m}/m$. The following example contrasts the flowshop-type robot move cycles with pure cycles.

Example 1. Let us consider two-machine cells with $\varepsilon = 1$, $\delta = 2$ and $P = 22$. In order to be able to compare flowshop-type cycles with pure cycles we also need the partial processing times of the parts on the machines for the flowshop-type robot move cycles. Let $P_1 = 14$ and $P_2 = 8$ so that $P_1 + P_2 = P = 22$, where P_1 (P_2) denotes the processing time of each of the parts on the first (second) machine. Let us compare cycles $S2$ and $L_1U_2L_2U_1$ as defined previously. This pure cycle is named as C_2^2 according to the above notation. The Gantt charts in Fig. 2 depict these two cycles on the same time line. Since one repetition of the C_2^2 cycle produces two parts, the $S2$ cycle is repeated twice. The bold dashed line in the middle of the Gantt chart of the $S2$ cycle illustrates the point where the first repetition is completed and the second repetition is started. As it is seen, with the given parameters, $\Omega_{C_2^2} = T_{C_2^2}/2 = \frac{38}{2} = 19$, whereas $\Omega_{S2} = 26$. This

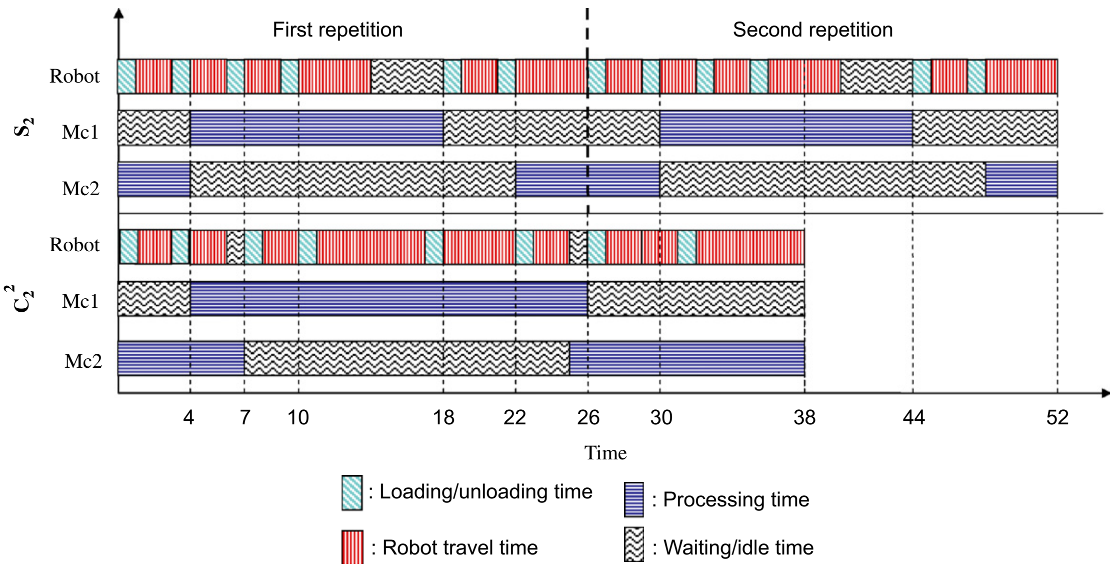

Fig. 2. Gantt charts for the $S2$ and the C_2^2 cycles.

corresponds to 27% reduction in the unit cycle time when a pure cycle is used instead of an optimum classical flowshop robot move cycle for this particular problem.

In the next section we will present a mathematical programming formulation for the problem and present the solution procedure.

3. Solution procedure

We can formulate this problem as a travelling salesman problem (TSP), where each of the activities $L_1, \ldots, L_m$, $U_1, \ldots, U_m$ represents a city to be visited. Let $[c_{lk}]$ for $l, k \in \mathscr{A}$ be the $2m \times 2m$ cost matrix for this TSP. In particular, for activities $l, k \in \mathscr{A}$, c_{lk} will correspond to the total travel time (including loading/unloading and transportation) in between the completion of activity l and the completion of activity k, assuming that activity k immediately succeeds activity l. Additionally, let w_j denote the robot waiting time in front of machine $j = 1, 2, \ldots, m$. Note that w_j is zero if the processing of the part is completed when the robot arrives in front of this machine to unload it. Otherwise, it is equivalent to the remaining processing time. Let t_l denote the time of completion of activity $l \in \mathscr{A}$. Also, let v_j denote the total activity time of the robot in between just after loading machine j (t_{L_j}) and arriving in front of machine j to unload it (let t_{a_j} denote this time epoch). Hence we can represent w_j as follows:

$$w_j = \max\{0, P - v_j\}. \tag{1}$$

t_{a_j} can be calculated using t_{U_j} as follows: after arriving in front of machine j to unload it (time t_{a_j}), the robot waits for the remaining processing time (w_j), unloads the machine (ε), transports the part to the output buffer ($(m+1-j)\delta$), drops the part (ε) (time t_{U_j}). This makes a total of $(2\varepsilon + (m+1-j)\delta + w_j)$. As a result, $t_{a_j} = t_{U_j} - (2\varepsilon + (m+1-j)\delta + w_j)$.

Calculation of v_j differs according to the respective order of the L_j and U_j in the activity sequence representing the pure cycle. If L_j precedes U_j in the activity sequence of a cycle, then as depicted in Fig. 3A, v_j can be calculated to be $t_{U_j} - (2\varepsilon + (m+1-j)\delta + w_j) - t_{L_j}$. Otherwise, as depicted in Fig. 3B, $v_j = T - (t_{L_j} - t_{U_j} + (2\varepsilon + (m+1-j)\delta + w_j))$. As a result, we have the following:

$$w_j = \begin{cases} \max\{0, P - (t_{U_j} - (2\varepsilon + (m+1-j)\delta + w_j) - t_{L_j})\} & \text{if } L_j \text{ precedes } U_j, \\ \max\{0, P - (T - (t_{L_j} - t_{U_j} + (2\varepsilon + (m+1-j)\delta + w_j)))\} & \text{otherwise.} \end{cases} \tag{2}$$

Note that, if U_j immediately succeeds L_j, then after loading a part to machine j, the robot waits in front of the machine to finish the processing of the part. Hence, the waiting time in such a case is equal to P. Since the t_l values are

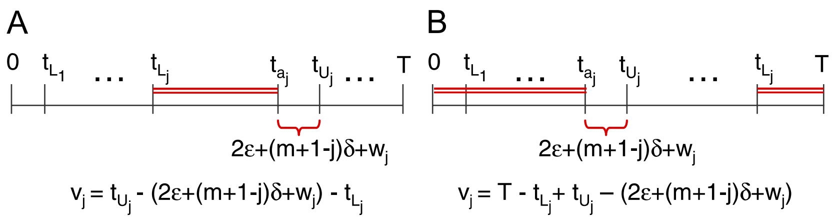
A
0
t_{L_1}
t_{L_j}
t_{a_j}
t_{U_j}
T
$2\varepsilon+(m+1-j)\delta+w_j$
$v_j = t_{U_j} - (2\varepsilon+(m+1-j)\delta+w_j) - t_{L_j}$
B
0
t_{L_1}
t_{a_j}
t_{U_j}
t_{L_j}
T
$2\varepsilon+(m+1-j)\delta+w_j$
$v_j = T - t_{L_j} + t_{U_j} - (2\varepsilon+(m+1-j)\delta+w_j)$

Fig. 3. Representation of the waiting times.

variables which depend on the sequence of the robot move activities, the waiting times are also variables. In order to determine c_{lk} for $l, k \in \mathscr{A}$, let us consider the four possibilities regarding the activity sequences l and k:

(1) L_j immediately succeeds L_i, $i \neq j$: Just after completing L_i, the robot is in front of machine i. It travels to the input buffer to take another part ($i\delta$), takes a part from the input buffer (ε), travels to machine j ($j\delta$) and loads it (ε). In particular, $c_{L_iL_j} = 2\varepsilon + (i+j)\delta$.
(2) U_j immediately succeeds L_i: The robot travels from machine i to machine j ($|i-j|\delta$), if necessary waits in front of the machine to finish the processing of the part (w_j), unloads the machine (ε), travels to the output buffer ($(m+1-j)\delta$), drops the part (ε). Hence, $c_{L_iU_j} = 2\varepsilon + (m+1-j+|i-j|)\delta$.
(3) L_j immediately succeeds U_i: Just after completing U_i, the robot is in front of the output buffer. It travels to the input buffer ($(m+1)\delta$), takes a part (ε), travels to machine j ($j\delta$) and loads it (ε). In other words, $c_{U_iL_j} = 2\varepsilon + (m+1+j)\delta$.
(4) U_j immediately succeeds U_i, $i \neq j$: The robot travels from the output buffer to machine j ($(m+1-j)\delta$), waits if necessary (w_j), unloads the machine (ε), travels to the output buffer ($(m+1-j)\delta$), drops the part (ε). This makes $c_{U_iU_j} = 2\varepsilon + 2(m+1-j)\delta$.

Let d_{lk} denote the total time from the completion of activity l to the completion of activity k, $l, k \in \mathscr{A}$. The distance from any activity, $l \in \mathscr{A}$, to a loading activity, L_j, $j = 1, \ldots, m$, denoted by d_{lL_j}, does not contain a waiting time in front of machine j. However, the distance from any activity, $l \in \mathscr{A}$, to an unloading activity, U_j, $j = 1, \ldots, m$, denoted by d_{lU_j}, requires a waiting time in front of machine j, w_j. Therefore, we have the following:

$$d_{lL_j} = c_{lL_j}, \quad \forall l \in \mathscr{A}, \quad j = 1, 2, \ldots, m \quad \text{and}$$
$$d_{lU_j} = c_{lU_j} + w_j, \quad \forall l \in \mathscr{A}, \quad j = 1, 2, \ldots, m.$$

As a result, the matrix composed of the d_{lk} values appears to be as follows:

	L_j	U_j
L_i	$2\varepsilon + (i+j)\delta$	$2\varepsilon + (m+1-j+\|i-j\|)\delta + w_j$
U_i	$2\varepsilon + (m+1+j)\delta$	$2\varepsilon + 2(m+1-j)\delta + w_j$

The following is an example of a distance matrix for a three-machine cell:

	L_1	L_2	L_3	U_1	U_2	U_3
L_1	–	$2\varepsilon+3\delta$	$2\varepsilon+4\delta$	$2\varepsilon+3\delta+P$	$2\varepsilon+3\delta+w_2$	$2\varepsilon+3\delta+w_3$
L_2	$2\varepsilon+3\delta$	–	$2\varepsilon+5\delta$	$2\varepsilon+4\delta+w_1$	$2\varepsilon+2\delta+P$	$2\varepsilon+2\delta+w_3$
L_3	$2\varepsilon+4\delta$	$2\varepsilon+5\delta$	–	$2\varepsilon+5\delta+w_1$	$2\varepsilon+3\delta+w_2$	$2\varepsilon+\delta+P$
U_1	$2\varepsilon+5\delta$	$2\varepsilon+6\delta$	$2\varepsilon+7\delta$	–	$2\varepsilon+4\delta+w_2$	$2\varepsilon+2\delta+w_3$
U_2	$2\varepsilon+5\delta$	$2\varepsilon+6\delta$	$2\varepsilon+7\delta$	$2\varepsilon+6\delta+w_1$	–	$2\varepsilon+2\delta+w_3$
U_3	$2\varepsilon+5\delta$	$2\varepsilon+6\delta$	$2\varepsilon+7\delta$	$2\varepsilon+6\delta+w_1$	$2\varepsilon+4\delta+w_2$	–

As can be observed, this time matrix is not solely composed of fixed parameters as in the classical TSP, but it also consists of variables such as w_i's. Let x_{lk} be the binary variable denoting whether activity l immediately precedes activity k or not in the best pure cycle. Then the following formulation determines the pure cycle with the minimum cycle time value:

$$\text{Minimize} \quad T \tag{3}$$

$$\text{Subject to} \quad t_k \geqslant t_l + d_{lk}x_{lk} - (1-x_{lk})M, \quad \forall l,k \in \mathscr{A}, \ l \neq k, \ k \neq L_1, \tag{4}$$

$$T \geqslant t_l + x_{lL_1}(d_{lL_1} - (2\varepsilon + \delta)) - (1 - x_{lL_1})M, \quad \forall l \in \mathscr{A}, \tag{5}$$

$$\sum_k x_{lk} = 1, \quad \forall l \in \mathscr{A}, \ l \neq k, \tag{6}$$

$$\sum_l x_{lk} = 1, \quad \forall k \in \mathscr{A}, \ k \neq l, \tag{7}$$

$$x_{lk} \in \{0,1\}, \quad \forall l,k \in \mathscr{A}. \tag{8}$$

The formulation above minimizes the cycle time. Constraint set (4) guarantees that, if activity k immediately follows activity l, then the completion of activity k is greater than or equal to the completion time of activity l plus the time from the completion time of activity l to the completion time of activity k. In this constraint, M denotes a big number which must at least be equal to the cycle time, T, so that if activity k does not immediately follow activity l, i.e., when $x_{lk} = 0$, then this constraint becomes redundant. We can determine a worst case value for the cycle time and use this value as M. In order to determine the worst case value we will consider the three components of the cycle time separately. Namely, the total loading/unloading time component, the total travelling time component and the total waiting time component. Since all parts are taken from the input buffer, loaded to any one of the machines, unloaded after the processing is completed and dropped to the output buffer, the total loading/unloading time is identical for all feasible pure cycles and is equal to $4m\varepsilon$. The maximum waiting time required for one machine is equal to its processing time, P. For producing m parts, the total waiting time is equal to mP. Finally, for each part, the robot transports the part from the input buffer to machine i, $i = 1, 2, \ldots, m$ ($i\delta$), from machine i to the output buffer ($(m+1-i)\delta$) and, in order to take another part, the robot must travel from the output buffer to any one of the machines. Maximum travel time from the output buffer is to the input buffer which is equal to ($(m+1)\delta$). Hence, for m machines, the maximum total travel time cannot be greater than $2m(m+1)\delta$. As a result, we can use as M value any value which is at least as large as $4m\varepsilon + 2m(m+1)\delta + mP$.

The second constraint states that the cycle time must be greater than or equal to the completion time of the last activity in the sequence plus the return time from the position of the robot just after completing this last activity to the input buffer (initial state of the system) denoted by $d_{lL_1} - (2\varepsilon + \delta)$ in the constraint. If the last activity is a loading activity, L_i, then $d_{L_iL_1} - (2\varepsilon + \delta) = i\delta$ and if it is an unloading activity, U_i, then $d_{U_iL_1} - (2\varepsilon + \delta) = (m+1)\delta$. The last two constraints are the classical assignment constraints of a TSP formulation.

Note that this formulation is a mixed integer nonlinear programming formulation. $d_{lk}x_{lk}$ multiplication in constraint (4) is the cause of nonlinearity. In order to linearize this, we consider loading and unloading activities separately. We know that $d_{lL_j}x_{lL_j} = c_{lL_j}x_{lL_j}$. However, $d_{lU_j}x_{lU_j} = c_{lU_j}x_{lU_j} + w_j x_{lU_j}$. Let us introduce a new variable, $y_{lU_j} = w_j x_{lU_j}$. Then we can replace constraint (4) with the following constraints:

$$t_{L_j} \geqslant t_l + c_{lL_j}x_{lL_j} - (1 - x_{lL_j})M, \quad \forall l \in \mathscr{A}, \ \forall j \in [1,\ldots,m], \ l \neq L_j, \ j \neq 1, \tag{9}$$

$$t_{U_j} \geqslant t_l + c_{lU_j}x_{lU_j} + y_{lU_j} - (1 - x_{lU_j})M, \quad \forall l \in \mathscr{A}, \ \forall j \in [1,\ldots,m], \ l \neq U_j, \tag{10}$$

$$y_{lU_j} \geqslant w_j - M(1 - x_{lU_j}), \quad \forall l \in \mathscr{A}, \ \forall j \in [1,\ldots,m], \tag{11}$$

$$y_{lU_j} \leqslant w_j + M(1 - x_{lU_j}), \quad \forall l \in \mathscr{A}, \ \forall j \in [1,\ldots,m], \tag{12}$$

$$y_{lU_j} \leqslant M x_{lU_j}, \quad \forall l \in \mathscr{A}, \ \forall j \in [1,\ldots,m], \tag{13}$$

$$y_{lU_j} \geqslant 0, \quad \forall l \in \mathscr{A}, \ \forall j \in [1,\ldots,m]. \tag{14}$$

The set of constraints (11)–(14) together force that if any activity l is immediately followed by an unloading activity $U_j (x_{lU_j} = 1)$, then $y_{lU_j} = w_j$. For the big number M in these constraints, we can use the same value determined already for constraint (4). If $x_{lU_j} = 0$, then the set of constraints (11)–(12) becomes redundant and (13) forces $y_{lU_j} = 0$.

Additionally, the max term in Eq. (1) can be linearized with the inclusion of the following constraints:

$$w_j \geqslant 0, \quad \forall j, \tag{15}$$

$$w_j \geqslant P - v_j, \quad \forall j. \tag{16}$$

Furthermore, since v_j has two alternatives as shown in Eq. (2), we can write these as the following set of constraints:

$$v_j \geqslant t_{U_j} - t_{L_j} - (2\varepsilon + (m+1-j)\delta + w_j) - Mz_j, \quad \forall j, \tag{17}$$

$$v_j \leqslant t_{U_j} - t_{L_j} - (2\varepsilon + (m+1-j)\delta + w_j) + Mz_j, \quad \forall j, \tag{18}$$

$$v_j \geqslant T - (t_{L_j} - t_{U_j} + (2\varepsilon + (m+1-j)\delta + w_j)) - M(1-z_j), \quad \forall j, \tag{19}$$

$$v_j \leqslant T - (t_{L_j} - t_{U_j} + (2\varepsilon + (m+1-j)\delta + w_j)) + M(1-z_j), \quad \forall j, \tag{20}$$

$$v_j \leqslant T, \quad \forall j, \tag{21}$$

$$z_j \in \{0, 1\}, \quad \forall j. \tag{22}$$

As a result we have the following mixed integer linear programming formulation:

Minimize T
Subject to (5)–(22).

The TSP is a well-known NP-hard problem. The formulation above 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. In our limited computational study on 20 randomly generated problems, the average CPU time required to solve problems with four machines was 7.72 s using CPLEX 9.0 commercial solver, whereas with five machines it was 1866.7 s. We only performed a single run with six machines, and had to cease the run after 805184.4 s with a 7.33% optimality gap. Consequently, we focused our attention on two specific pure cycles from the huge number of pure cycles of an m-machine cell and showed that they perform very effectively compared to others. We will determine the regions of optimality for these two cycles. For the remaining regions, we will derive their worst case performance bounds. Let us first define these two cycles.

Definition 4. C_1^m is the robot move cycle in an m-machine robotic cell with the following activity sequence: $L_1L_mU_{m-1}L_{m-1}U_{m-2}L_{m-2}\ldots U_2L_2U_1U_m$.

Definition 5. C_2^m is the robot move cycle in an m-machine robotic cell with the following activity sequence: $L_1U_mL_mU_{m-1}L_{m-1}\ldots U_2L_2U_1$.

In the first pure cycle, C_1^m, in the initial state of the system, the machines 1 and m are idle and the rest of the machines 2 to $m-1$ are already loaded with a part. In cycle C_2^m, only the first machine is idle and the remaining ones are busy in the initial state. In the following two lemmas we derive the cycle times of these two cycles.

Lemma 1. *The cycle time of C_1^m is the following*:

$$T_{C_1^m} = 4m\varepsilon + 2m(m+1)\delta + \max\{0, P - ((4m-6)\varepsilon + 2(m^2-2)\delta)\}. \tag{23}$$

Proof. For the clarity of the presentation, the proof is placed in Appendix.

Lemma 2. *The cycle time of C_2^m is the following*:

$$T_{C_2^m} = 4m\varepsilon + 2((m+1)^2-2)\delta + \max\{0, P - ((4m-4)\varepsilon + 2(m-1)(m+2)\delta)\}. \tag{24}$$

Proof. For the proof please refer to Appendix.

The following theorem provides a lower bound for the cycle time of the pure cycles. We will use this bound to prove the dominance of these two cycles.

Theorem 1. *For an m-machine robotic cell, the cycle time of any pure cycle is no less than*

$$\underline{T_{\text{pure}}} = \max\{4m\varepsilon + 2m(m+1)\delta, 4\varepsilon + (2m+2)\delta + P\}. \tag{25}$$

Proof. The first argument results from the following observation: any part to be produced with one of the pure cycles is taken from the input buffer (ε), loaded to one of the machines and unloaded after the processing is completed (2ε) and dropped to the output buffer (ε), which makes a total of 4ε. Since a cycle produces m parts, the total time amounts to $4m\varepsilon$. Also for each part, the robot travels from the input buffer to output buffer and returns back either to take another part or to complete the cycle which makes $2(m+1)\delta$. For all of the m parts this totals to $2m(m+1)\delta$. On the other hand, the second argument of the lower bound is the minimum time between two consecutive loadings of any machine. After loading any machine, the minimum time required before the robot can unload it is P. Let us consider machine i without loss of generality. The robot unloads the machine (ε), travels to the output buffer ($(m+1-i)\delta$), drops the part (ε), travels to the input buffer ($(m+1)\delta$), takes a part (ε), brings the part to machine i ($i\delta$) and loads the machine (ε). This makes a total of $4\varepsilon + (2m+2)\delta + P$. □

In the following theorem, we compare C_1^m and C_2^m cycles with each other.

Theorem 2. *If $P < (4m-6)\varepsilon + 2(m^2+m-3)\delta$, then C_1^m dominates C_2^m; else if $P > (4m-6)\varepsilon + 2(m^2+m-3)\delta$, then C_2^m dominates C_1^m. If $P = (4m-6)\varepsilon + 2(m^2+m-3)\delta$, then both cycles perform equally well.*

Proof. We will compare the cycle times of these two cycles in the following cases:

1. If $P \leqslant (4m-6)\varepsilon + 2(m^2-2)\delta$, then $T_{C_1^m} = 4m\varepsilon + 2m(m+1)\delta \leqslant 4m\varepsilon + 2((m+1)^2-2)\delta = T_{C_2^m}$.
2. If $(4m-6)\varepsilon + 2(m^2-2)\delta < P \leqslant (4m-4)\varepsilon + 2(m-1)(m+2)\delta$, then $T_{C_1^m} = 6\varepsilon + (2m+4)\delta + P$. If $P = (4m-6)\varepsilon + 2(m^2+m-3)\delta$, then $T_{C_1^m} = T_{C_2^m} = 4m\varepsilon + 2((m+1)^2-2)\delta$. Hence, if $P < (4m-6)\varepsilon + 2(m^2+m-3)\delta$ that means $T_{C_1^m} < T_{C_2^m}$. Else if $P > (4m-6)\varepsilon + 2(m^2+m-3)\delta$, then $T_{C_1^m} > T_{C_2^m}$.
3. If $P > (4m-4)\varepsilon + 2(m-1)(m+2)\delta$, then $T_{C_1^m} = 6\varepsilon + (2m+4)\delta + P \geqslant 4\varepsilon + (2m+2)\delta + P = T_{C_2^m}$.

This completes the proof. □

In the following theorem, we determine the regions of optimality of C_1^m and C_2^m cycles.

Theorem 3. 1. *If $P \leqslant (4m-6)\varepsilon + 2(m^2-2)\delta$, then C_1^m is the optimal pure cycle.*
2. *If $P \geqslant (4m-4)\varepsilon + 2(m-1)(m+2)\delta$, then C_2^m is the optimal pure cycle.*

Proof. 1. If $P \leqslant (4m-6)\varepsilon + 2(m^2-2)\delta$,

$$T_{C_1^m} = 4m\varepsilon + 2m(m+1)\delta = \underline{T_{\text{pure}}}.$$

2. If $P \geqslant (4m-4)\varepsilon + 2(m-1)(m+2)\delta$,

$$T_{C_2^m} = 4\varepsilon + (2m+2)\delta + P = \underline{T_{\text{pure}}}. \quad □$$

In the following lemmas we consider the remaining region where $(4m-6)\varepsilon + 2(m^2-2)\delta < P < (4m-4)\varepsilon + 2(m-1)(m+2)\delta$, and derive worst case performances of the C_1^m and C_2^m cycles. Let T_m^* denote the optimal pure cycle.

Lemma 3. *If $(4m-6)\varepsilon + 2(m^2-2)\delta < P \leqslant (4m-6)\varepsilon + 2(m^2+m-3)\delta$, then $T_{C_1^m} \leqslant (1 + 1/(2m)) \cdot T_m^*$.*

Proof. For $P > (4m-6)\varepsilon + 2(m^2-2)\delta$, $T_{C_1^m} = 6\varepsilon + (2m+4)\delta + P$. Then, we have two cases:

1. If $P \leqslant (4m-4)\varepsilon + 2(m^2-1)\delta$, from Eq. (25), $\underline{T_{\text{pure}}} = 4m\varepsilon + 2m(m+1)\delta$. Hence, we have the following:

$$\frac{T_{C_1^m}}{T_m^*} \leqslant \frac{6\varepsilon + (2m+4)\delta + P}{4m\varepsilon + 2m(m+1)\delta}.$$

Since $P \leqslant (4m-4)\varepsilon + 2(m^2-1)\delta$,

$$\frac{T_{C_1^m}}{T_m^*} \leqslant \frac{(4m+2)\varepsilon + (2m^2+2m+2)\delta}{4m\varepsilon + 2m(m+1)\delta} \leqslant 1 + \frac{1}{2m} - \frac{(m-1)\delta}{4m\varepsilon + 2m(m+1)\delta} \leqslant 1 + 1/(2m).$$

2. Else if $P > (4m-4)\varepsilon + 2(m^2-1)\delta$, from Eq. (25), $\underline{T}_{\text{pure}} = 4\varepsilon + (2m+2)\delta + P$. Hence, we have the following:

$$\frac{T_{C_1^m}}{T_m^*} \leqslant \frac{6\varepsilon + (2m+4)\delta + P}{4\varepsilon + (2m+2)\delta + P} = 1 + \frac{2(\varepsilon+\delta)}{4\varepsilon + (2m+2)\delta + P}.$$

Since $P > (4m-4)\varepsilon + 2(m^2-1)\delta$,

$$\frac{T_{C_1^m}}{T_m^*} < 1 + \frac{\varepsilon+\delta}{2m\varepsilon + m(m+1)\delta} < 1 + \frac{1}{2m} - \frac{(m-1)\delta}{2(2m\varepsilon + m(m+1)\delta)} < 1 + 1/(2m). \qquad \square$$

Lemma 4. *If* $(4m-6)\varepsilon + 2(m^2+m-3)\delta \leqslant P < (4m-4)\varepsilon + 2(m-1)(m+2)\delta$, *then* $T_{C_2^m} < (1+1/m) \cdot T_m^*$.

Proof. For $P < (4m-4)\varepsilon + 2(m-1)(m+2)\delta$, $T_{C_2^m} = 4m\varepsilon + (2m^2+4m-2)\delta$. Then, we have two cases:

1. If $P \leqslant (4m-4)\varepsilon + 2(m^2-1)\delta$, from Eq. (25), $\underline{T}_{\text{pure}} = 4m\varepsilon + 2m(m+1)\delta$. Hence, we have the following:

$$\frac{T_{C_2^m}}{T_m^*} \leqslant \frac{4m\varepsilon + (2m^2+4m-2)\delta}{4m\varepsilon + 2m(m+1)\delta} = 1 + \frac{(m-1)\delta}{2m\varepsilon + m(m+1)\delta}$$
$$= 1 + \frac{1}{m} - \frac{2(\varepsilon+\delta)}{2m\varepsilon + m(m+1)\delta}.$$

Hence,

$$\frac{T_{C_2^m}}{T_m^*} < 1 + 1/m.$$

2. Else if $P > (4m-4)\varepsilon + 2(m^2-1)\delta$, from Eq. (25), $\underline{T}_{\text{pure}} = 4\varepsilon + (2m+2)\delta + P$. Hence, we have the following:

$$\frac{T_{C_2^m}}{T_m^*} \leqslant \frac{4m\varepsilon + (2m^2+4m-2)\delta}{4\varepsilon + (2m+2)\delta + P}.$$

Since $P > (4m-4)\varepsilon + 2(m^2-1)\delta$,

$$\frac{T_{C_2^m}}{T_m^*} < \frac{4m\varepsilon + (2m^2+4m-2)\delta}{4m\varepsilon + (2m^2+2m)\delta} = 1 + \frac{(2m-2)\delta}{4m\varepsilon + (2m^2+2m)\delta}$$
$$= 1 + \frac{1}{m} - \frac{4(\varepsilon+\delta)}{4m\varepsilon + (2m^2+2m)\delta} < 1 + 1/m. \qquad \square$$

4. Managerial insight

In this section we will compare the flowshop-type robot move cycles considered in the robotic cell scheduling literature with the pure cycles considered in this study. Recall that in a flowshop-type robot move cycle, a part starting from the first machine visits all machines in the same sequence where the last operation is performed by the mth machine. A flowshop-type cycle can be defined as an "n-unit cycle" if one repetition produces n parts. This means that all machines are loaded and unloaded exactly n times and each machine performs some specific operations on the parts. In these cycles, robot makes a loaded travel only between two consecutive machines. However, in pure cycles, a part is completely processed by only one machine. Robot makes a loaded move only between the input/output buffer and a machine but not between two machines. In one repetition of a pure cycle all machines are loaded and unloaded exactly once. Since there are m machines in the cell, one repetition of a pure cycle produces exactly m parts.

Interestingly, pure cycles are used extensively in industry, not because they are proved to be optimal but because they are very practical, easy to understand and implement. The main question was to select the best pure cycle among the many feasible pure cycle alternatives that were addressed in the previous sections. In what follows below, we prove that the pure cycles are not only simple and practical but also dominate all classical flowshop-type robot move cycles. As a final remark, in any pure robot move cycle each part is loaded and unloaded only once, which means less gaging, probably one of the important reasons why this cycle is preferred in practice.

Let $\underline{T_{fs(m)}}$ denote the lower bound of the cycle times of the flowshop-type robot move cycles of an m-machine robotic cell. The following theorem is proved by Gultekin et al. [7].

Theorem 4. (Gultekin et al. [7]) *For an m- machine flowshop-type robotic cell, the cycle time of any n-unit cycle is no less than*

$$\underline{T_{fs(m)}} = \max\{2m(m+1)(\varepsilon+\delta) + \min\{P, \delta\}, 4m\varepsilon + 4m\delta + (P)\}. \tag{26}$$

With the following theorem, we prove that pure cycles dominate the flowshop-type robot move cycles.

Theorem 5. *Pure cycle C_1^m dominates all flowshop-type robot move cycles.*

Proof. In order to prove this theorem we will compare the lower bound of the cycle times of the flowshop-type robot move cycles with the cycle time of the C_1^m cycle and prove that even only the C_1^m cycle dominates the flowshop-type robot move cycles. The cycle time of the C_1^m cycle is given in Eq. (23) and the lower bound of the cycle time of flowshop-type robot move cycles is given in Eq. (26). We will compare these in the following cases:

1. If $P \leqslant \delta$, then

$$T_{C_1^m} = 4m\varepsilon + 2m(m+1)\delta < (2m^2+2m)\varepsilon + 2m(m+1)\delta + P = \underline{T_{fs(m)}}.$$

2. If $\delta < P \leqslant (4m-6)\varepsilon + 2(m^2-2)\delta$, then $T_{C_1^m} = 4m\varepsilon + 2m(m+1)\delta$. We have to consider the following cases with respect to $\underline{T_{fs(m)}}$:

2.1. If $P \leqslant 2m(m-1)\varepsilon + (2m^2-2m+1)\delta$, then

$$\underline{T_{fs(m)}} = 2m(m+1)\varepsilon + (2m^2+2m+1)\delta > 4m\varepsilon + 2m(m+1)\delta = T_{C_1^m}.$$

2.2. If $P > 2m(m-1)\varepsilon + (2m^2-2m+1)\delta$, then $\underline{T_{fs(m)}} = 4m\varepsilon + 4m\delta + P$. Since $P > 2m(m-1)\varepsilon + (2m^2-2m+1)\delta$, we have

$$\begin{aligned}\underline{T_{fs(m)}} &= 4m\varepsilon + 4m\delta + P > 4m\varepsilon + 4m\delta + 2m(m-1)\varepsilon + (2m^2-2m+1)\delta \\ &= (2m^2+2m)\varepsilon + (2m^2+2m+1) > 4m\varepsilon + 2m(m+1)\delta = T_{C_1^m}.\end{aligned}$$

3. If $P > (4m-6)\varepsilon + 2(m^2-2)\delta$, then $T_{C_1^m} = 6\varepsilon + (2m+4)\delta + P$. We have to consider the following cases with respect to $\underline{T_{fs(m)}}$:

3.1. If $P \leqslant 2m(m-1)\varepsilon + (2m^2-2m+1)\delta$, then

$$\begin{aligned}T_{C_1^m} &= 6\varepsilon + (2m+4)\delta + P \leqslant (2m^2-2m+6)\varepsilon + (2m^2+5)\delta \\ &= 2m(m+1)\varepsilon - (4m-6)\varepsilon + (2m^2+2m+1)\delta - (2m-4)\delta.\end{aligned}$$

For $m \geqslant 2$,

$$\begin{aligned}T_{C_1^m} &\leqslant 2m(m+1)\varepsilon - (4m-6)\varepsilon + (2m^2+2m+1)\delta - (2m-4)\delta \\ &< 2m(m+1)\varepsilon + (2m^2+2m+1)\delta = \underline{T_{fs(m)}}.\end{aligned}$$

3.2. If $P > 2m(m-1)\varepsilon + (2m^2-2m+1)\delta$, then

$$T_{C_1^m} = 6\varepsilon + (2m+4)\delta + P = 4m\varepsilon - (4m-6)\varepsilon + 4m\delta - (2m-4)\delta + P.$$

For $m \geqslant 2$,

$$4m\varepsilon - (4m-6)\varepsilon + 4m\delta - (2m-4)\delta + P < 4m\varepsilon + 4m\delta + P = \underline{T_{fs(m)}}.$$

This completes the proof. □

This theorem proves that the set of pure cycles dominates all flowshop-type robot move cycles. As a result, if the considered cell is an FMC consisting of CNC machines, then assuming the system to be a flowshop-type system, as it is done in the current literature, results in suboptimal solutions. By fully utilizing the flexibility of the machines, the throughput rate of these systems can be increased even further. Furthermore, with the reduced cycle times (increased throughput), our results enable the justification of additional tool inventories that will be incurred when loading a copy of every required tool to each one the tool magazines (this might also necessitate a larger tool magazine).

Another factor affecting the throughput rates of such cells is the design of the cell. Most of the earlier research on this considered operational problems such as finding the part input sequence and the robot move sequence. However, the number of the machines in a cell can be considered as a decision variable and, for given parameters such as the processing times on the machines, the robot travel times and the loading/unloading times of the machines, the optimum number of machines that minimizes the cycle time can be determined. The results of such analysis are useful for determining the equipment requirements and the designs of such cells. In the sequel, we determine the optimum number of machines for the C_1^m and C_2^m cycles separately that minimizes the per unit cycle time, Ω_S.

Let us first consider the C_1^m cycle. The per unit cycle time for the C_1^m cycle can be found by dividing the cycle time given in Eq. (23) by m, which can be written as follows:

$$\Omega_{C_1^m} = \max\{4\varepsilon + (2m+2)\delta, 2\delta + (6\varepsilon + 4\delta + P)/m\}. \tag{27}$$

The following theorem determines the optimal number of machines to be used with the C_1^m cycle for given P, ε and δ.

Theorem 6. *The optimal number of machines*, m^*, *for the* C_1^m *cycle is one of the two integers* $\lfloor(1/\delta)(\varepsilon^2 + (\delta P)/2 + 6\varepsilon\delta + 8\delta^2 - \varepsilon)\rfloor$ *or* $\lfloor(1/\delta)(\varepsilon^2 + (\delta P)/2 + 6\varepsilon\delta + 8\delta^2 - \varepsilon)\rfloor + 1$.

Proof. The first argument of the max function in Eq. (27), $4\varepsilon + (2m+2)\delta$, is linear with respect to m and the second argument, $2\delta + (6\varepsilon + 4\delta + P)/m$, is convex with respect to m. Since the maximum of two convex functions is also a convex function, $\Omega_{C_1^m}$ is convex with respect to m. For such functions, the minimizer of the max function is either the minimizer of one of the two functions or the intersection point of the two arguments of the max function. The first argument is a linear increasing function with respect to m, for which the minimum is attained at $m = 0$. However, the number of machines in the cell must be at least one. The minimum of the second argument is attained for $m \to \infty$. However, since the first argument is an increasing function, as $m \to \infty$, $4\varepsilon + (2m+2)\delta \to \infty$. Hence, this point cannot be a minimizer of the max function. Let us now consider the intersection point of the two arguments:

$$4\varepsilon + (2m+2)\delta = 1/m(6\varepsilon + (2m+4)\delta + P).$$

After a few manipulations we get the following:

$$2\delta m^2 + 4\varepsilon m - (6\varepsilon + 4\delta + P) = 0.$$

This equation has two roots, one of which is negative. However, since the decision variable is the number of machines, we take the positive root:

$$\frac{\sqrt{\varepsilon^2 + (\delta P)/2 + 6\varepsilon\delta + 8\delta^2} - \varepsilon}{\delta}.$$

Since the number of machines cannot be fractional, we take the smallest integer larger than and the largest integer smaller than this value. The optimal solution is found by comparing these with each other and selecting the one which gives the smallest cycle time value. □

Similarly, the following theorem determines the optimal number of machines for the C_2^m cycle. The per unit cycle time of the C_2^m cycle can be found by dividing $T_{C_2^m}$ given in Eq. (24) by m as follows:

$$\Omega_{C_2^m} = \max\{4\varepsilon + 4\delta + 2m\delta - (2\delta)/m, 2\delta + (1/m)(4\varepsilon + 2\delta + P)\}. \tag{28}$$

Theorem 7. *The optimal number of machines, m^*, for the C_2^m cycle is one of the two integers $\lfloor (1/2\delta)(4\varepsilon^2 + 2\delta P + 12\varepsilon\delta + \delta^2 - 2\varepsilon - \delta)\rfloor$ or $\lfloor (1/2\delta)(4\varepsilon^2 + 2\delta P + 12\varepsilon\delta + \delta^2 - 2\varepsilon - \delta)\rfloor + 1$.*

Proof. Both arguments of the max function are convex and maximum of these is also convex. The minimizer of the first argument is $m = \sqrt{-1}$, which is not a real value. On the other hand, the minimizer of the second argument is $m \to \infty$. However, as $m \to \infty$, $4\varepsilon + 4\delta + 2m\delta - (2\delta)/m \to \infty$. Thus, this cannot be a minimizer of the cycle time. As a consequence, the minimizer is the intersection point of the two arguments of the max function.

$$4\varepsilon + 4\delta + 2m\delta - (2\delta)/m = 2\delta + (1/m)(4\varepsilon + 2\delta + P).$$

After a few manipulations we get the following:

$$2\delta m^2 + (4\varepsilon + 2\delta)m - (4\varepsilon + P) = 0.$$

This equation has two roots, one of which is negative. For the number of machines, we take the positive root:

$$\frac{\sqrt{4\varepsilon^2 + 2\delta P + 12\varepsilon\delta + \delta^2} - 2\varepsilon - \delta}{2\delta}.$$

Since the number of machines cannot be fractional, we take the smallest integer larger than this and the largest integer smaller than this value. The optimal solution is found by comparing these with each other and selecting the one which gives the smallest cycle time value. □

Next section concludes this study and suggests some future research directions.

5. Conclusion

In this study, an m-machine robotic cell used for metal cutting operations is considered. The machines used in such manufacturing cells are CNC machines which are highly flexible. As a consequence, each part is assumed to be composed of a number of operations and each machine is assumed to be capable of performing all of the required operations of each part. We investigated the productivity gain attained by the additional flexibility introduced by the CNC machines.

A new class of robot move cycles, namely the pure cycles, which resulted from the flexibility of the machines are defined. The problem is formulated as a TSP, where we have a special time matrix. Due to the extensive computational effort required to solve this formulation, we determined two specific pure cycles which perform effectively. We determined the regions of optimality for both of these cycles. For the remaining small region, we determined worst case bounds for both of these cycles. We also proved that the set of pure cycles dominates all flowshop-type robot move cycles. The results show that these proposed cycles are not only simple and practical but perform very efficiently as well. As a design problem, we considered the number of machines in a cell as a decision variable and determined the optimal number of machines for the two specific pure cycles. Extending the analysis to the multiple parts case can be considered as a future research direction. In such a case, the determination of the part input sequence must also be tackled which will certainly increase the complexity of the problem even further.

Acknowledgements

The authors would like to thank Prof. Gerd Finke from University Joseph Fourier, Leibniz-IMAG, France, for his helpful comments on an earlier draft of this paper. This work is partially supported by a grant from Turkish Academy of Science (TÜBA).

Appendix A. Derivation of the cycle times of C_1^m and C_2^m

Proof of Lemma 1. Let t_l be the completion time of activity $l \in \mathscr{A}$. More specifically, if $l = L_i$, then t_l is the time just after loading machine i. If $l = U_i$, then t_l is the time just after dropping the part to the output buffer. Then, for the cycle C_1^m, we have the following:

$$\begin{aligned}
t_{L_1} &= 2\varepsilon + \delta,\\
t_{L_m} &= t_{L_1} + 2\varepsilon + (m+1)\delta,\\
t_{U_i} &= t_{L_{(i+1)}} + 2\varepsilon + (m-i+2)\delta + w_i, \quad i = m-1, m-2, \ldots, 3, 2,\\
t_{L_i} &= t_{U_i} + 2\varepsilon + (m+i+1)\delta = t_{L_{i+1}} + 4\varepsilon + (2m+3)\delta + w_i, \quad i = m-1, m-2, \ldots, 3, 2,\\
t_{U_1} &= t_{L_2} + 2\varepsilon + (m+1)\delta + w_1,\\
t_{U_m} &= t_{U_1} + 2\varepsilon + 2\delta + w_m.
\end{aligned}$$

Since the cycle is completed after the robot returns back to the input buffer, the cycle time of the C_1^m cycle can be written as: $T_{C_1^m} = t_{U_m} + (m+1)\delta$. Considering the equations above, one can determine the following:

$$T_{C_1^m} = 4m\varepsilon + (2m^2 + 2m)\delta + w_1 + w_2 + \cdots + w_m,$$

where w_i denotes the waiting time in front of machine i. This can be represented as $w_i = \max\{0, P - v_i\}$, where v_i denotes the time between loading machine i and the arrival time of the robot in front the same machine to unload it. We can calculate these as follows: Let us first calculate v_1. Just after arriving in front of the first machine, the robot waits for the processing to be completed (w_1), unloads the machine (ε), travels to the output buffer ($m\delta$), drops the part (ε), travels to machine m (δ), waits for the remaining processing time (w_m), unloads the machine (ε), travels to the output buffer (δ), drops the part (ε), travels to the input buffer ($(m+1)\delta$), takes a part (ε), travels to the first machine (δ), loads it (ε). This makes a total of $6\varepsilon + (2m+4)\delta + w_1 + w_m$. Then, v_1 is equivalent to the remaining time of a cycle. That is,

$$v_1 = T_{C_1^m} - (6\varepsilon + (2m+4)\delta + w_1 + w_m) = (4m-6)\varepsilon + (2m^2 - 4)\delta + w_2 + \cdots + w_{m-1}.$$

Similarly,

$$v_m = T_{C_1^m} - (6\varepsilon + (2m+4)\delta + w_m) = (4m-6)\varepsilon + (2m^2 - 4)\delta + w_1 + w_2 + \cdots + w_{m-1}.$$

After arriving in front of machine i, $i = 2, 3, \ldots, m-1$, the robot waits for the remaining processing time (w_i), unloads the machine (ε), travels to the output buffer ($(m-i+1)\delta$), drops the part (ε), travels to the input buffer ($(m+1)\delta$), takes a part (ε), travels to machine i ($i\delta$), loads the machine (ε). This makes a total of $4\varepsilon + (2m+2)\delta + w_i$. Hence, we have the following:

$$v_i = T_{C_1^m} - (4\varepsilon + (2m+2)\delta + w_i) = (4m-4)\varepsilon + (2m^2 - 2)\delta + w_1 + \cdots + w_m - w_i, \quad i = 2, \ldots, m-1.$$

Note that $v_m = v_1 + w_1$ and $v_i = v_1 + w_1 + w_m - w_i + 2\varepsilon + 2\delta$, $i = 2, 3, \ldots, m-1$. We have two cases:

1. If $v_1 \geqslant P$, then $w_1 = 0$. Since $v_m \geqslant v_1$, then $w_m = 0$. $v_i = v_1 - w_i + 2\varepsilon + 2\delta$. Let us consider these two cases. First let $w_i = P - v_i > 0$. Then, $v_i = v_1 - P + v_i + 2\varepsilon + 2\delta \Rightarrow P = v_1 + 2\varepsilon + 2\delta$, which contradicts with $v_1 \geqslant P$. As a result, if $w_1 = 0$, then $w_i = 0$, for $i = 2, 3, \ldots, m$.
2. If $v_1 < P$, then $w_1 = P - v_1$. As a consequence, $v_m = v_1 + P - v_1 = P \Rightarrow w_m = 0$. From here, $v_i = v_1 + P - v_1 - w_i + 2\varepsilon + 2\delta = P - w_i + 2\varepsilon + 2\delta$. Let $w_i = P - v_i > 0$. Then, $v_i = P - P + v_i + 2\varepsilon + 2\delta = v_i + 2\varepsilon + 2\delta$, which is not possible. As a result, if $w_1 > 0$, then $w_i = 0$, for $i = 2, 3, \ldots, m$.

Hence, $w_1 + w_2 + \cdots + w_m = w_1$. As a result, $T_{C_1^m}$ is found to be as follows:

$$T_{C_1^m} = 4m\varepsilon + 2m(m+1)\delta + \max\{0, P - ((4m-6)\varepsilon + 2(m^2-2)\delta)\}. \qquad \square$$

Proof of Lemma 2. For the cycle C_2^m, we have the following:

$$\begin{aligned}
&t_{L_1} = 2\varepsilon + \delta,\\
&t_{U_m} = t_{L_1} + 2\varepsilon + (m)\delta + w_m,\\
&t_{U_i} = t_{L_{(i+1)}} + 2\varepsilon + (m-i+2)\delta + w_i, \quad i = m-1, m-2, \ldots, 2, 1,\\
&t_{L_i} = t_{U_i} + 2\varepsilon + (m+i+1)\delta, \quad i = m, m-1, \ldots, 3, 2.
\end{aligned}$$

Since the cycle is completed after the robot returns back to the input buffer, the cycle time of the C_2^m cycle can be found as: $T_{C_2^m} = t_{U_1} + (m+1)\delta$. Considering the above, one can determine the following:

$$T_{C_2^m} = 4m\varepsilon + (2m^2 + 4m - 2)\delta + w_1 + w_2 + \cdots + w_m.$$

In this cycle, after arriving in front of machine i, $i = 1, 2, \ldots, m$, to unload it, the robot waits for the processing to be completed (w_i), unloads the machine (ε), travels to the output buffer ($(m-i+1)\delta$), drops the part (ε), travels to the input buffer ($(m+1)\delta$), takes a part (ε), travels to machine i ($i\delta$), loads it (ε). This makes a total of $4\varepsilon + (2m+2)\delta + w_i$. Then, v_i is equivalent to the remaining time to complete the cycle. That is,

$$\begin{aligned}
v_i &= T_{C_2^m} - (4\varepsilon + (2m+2)\delta + w_i)\\
&= (4m-4)\varepsilon + 2(m-1)(m+2)\delta + w_1 + \cdots + w_m - w_i, \quad i = 1, \ldots, m.
\end{aligned}$$

Now let us consider the two possible cases that might arise:

1. If $P \leqslant \min_{i\in[1,\ldots,m]}\{v_i\}$, then $w_i = 0$, for $i = 1, 2, \ldots, m$.
2. If $\exists k \in [1, \ldots, m]$ such that $P > v_k$, then $w_k = P - v_1 = P - (4m-4)\varepsilon - 2(m-1)(m+2)\delta - \sum_{i\neq k} w_k$. Hence, $w_1 + w_2 + \cdots + w_m = P - (4m-4)\varepsilon - 2(m-1)(m+2)\delta$.

As a consequence, $w_1 + w_2 + \cdots + w_m = \max\{0, P - (4m-4)\varepsilon - 2(m-1)(m+2)\delta\}$ and the cycle time is as follows:

$$T_{C_2^m} = 4m\varepsilon + 2((m+1)^2 - 2)\delta + \max\{0, P - ((4m-4)\varepsilon + 2(m-1)(m+2)\delta)\}. \qquad \square$$

References

[1] Crama Y, Kats V, Van de Klundert J, Levner E. Cyclic scheduling in robotic flowshops. Annals of Operations Research 2000;96:97–124.

[2] Dawande M, Geismar HN, Sethi S, Sriskandarajah C. Sequencing and scheduling in robotic cells: recent developments. Journal of Scheduling 2005;8:387–426.

[3] Sethi SP, Sriskandarajah C, Sorger G, Blazewicz J, Kubiak W. Sequencing of parts and robot moves in a robotic cell. International Journal of Flexible Manufacturing Systems 1992;4:331–58.

[4] Crama Y, Van de Klundert J. Cyclic scheduling in 3-machine robotic flow shops. Journal of Scheduling 1999;4:35–54.

[5] Brauner N, Finke G. On cycles and permutations in robotic cells. Mathematical and Computer Modeling 2001;34:565–91.

[6] Akturk MS, Gultekin H, Karasan OE. Robotic cell scheduling with operational flexibility. Discrete Applied Mathematics 2005;145:334–48.

[7] Gultekin H, Akturk, MS, Karasan OE. Scheduling in robotic cells: process flexibility and cell layout. International Journal of Production Research 2007, in press.

[8] Gultekin H, Akturk MS, Karasan OE. Scheduling in a three-machine robotic flexible manufacturing cell. Computers and Operations Research 2007;34:2463–77.

[9] Hall NG, Potts CN, Sriskandarajah C. Parallel machine scheduling with a common server. Discrete Applied Mathematics 2000;102:223–43.

[10] Abdekhodaee AH, Wirth A, Gan HS. Equal processing and equal setup time cases of scheduling parallel machines with a single server. Computers and Operations Research 2004;31:1867–89.

[11] Gray AE, Seidmann A, Stecke KE. A synthesis of decision models for tool management in automated manufacturing. Management Science 1993;39:549–67.

[12] Crama Y, Van de Klundert J. Cyclic scheduling of identical parts in a robotic cell. Operations Research 1997;45:952–65.