SIAM REVIEW
Vol. 47, No. 4, pp. 709–721

Dominance of Cyclic Solutions and Challenges in the Scheduling of Robotic Cells*

Milind W. Dawande†
H. Neil Geismar‡
Suresh P. Sethi†

Abstract. We consider the problem of scheduling operations in bufferless robotic cells that produce identical parts. Maximizing the long-term average throughput of parts is an important problem in both theory and practice. We define an appropriate state space required to analyze this problem and show that cyclic schedules which repeat a fixed sequence of robot moves indefinitely are the only ones that need to be considered. For the different classes of robotic cells studied in the literature, we discuss the current state of knowledge with respect to cyclic schedules. Finally, we discuss the importance of two fundamental open problems concerning optimal cyclic schedules, special cases for which these problems have been solved, and attempts to solve the general case.

Key words. combinatorial optimization, robotics, discrete-time systems, scheduling theory

AMS subject classifications. 90C27, 68T40, 93C55, 90B35

DOI. 10.1137/S003614450444138X

1. Introduction.

The robotic cells considered in this paper consist of a number of machines served by a central robot. Parts are brought to an input station by an input conveyor and the finished parts are removed from an output station by an output conveyor. As in a classical flowshop [34], each part being processed passes successively from the input station, through the machines in a fixed sequence, and finally to the output station. Each machine performs a specific process on each part and can contain only one part at a time. There are no buffers for intermediate storage of parts at the machines, and any part in the cell is always either on one of the machines or in the robot arm. All intermachine transfers are handled by the robot arm.

A three-machine robotic cell is illustrated in Figure 1. After loading a part onto a machine, either the robot waits at the machine for it to finish its processing of the part, or it moves to another machine to unload a part as soon as that machine has finished processing it, or it moves to the input station to pick up a new part. Neither the machines nor the robot can be in possession of more than one part at any given time. This and the lack of internal buffers impose a *blocking* condition: a part cannot be removed from its current machine unless the next machine in the sequence

*Received by the editors February 20, 2004; accepted for publication (in revised form) October 4, 2004; published electronically October 31, 2005.
http://www.siam.org/journals/sirev/47-4/44138.html

†School of Management, University of Texas at Dallas, Dallas, TX 75083 (milind@utdallas.edu, sethi@utdallas.edu).
‡College of Business, Prairie View A & M University, P. O. Box 519, Prairie View, TX 77446-0519 (neil_geismar@pvamu.edu).

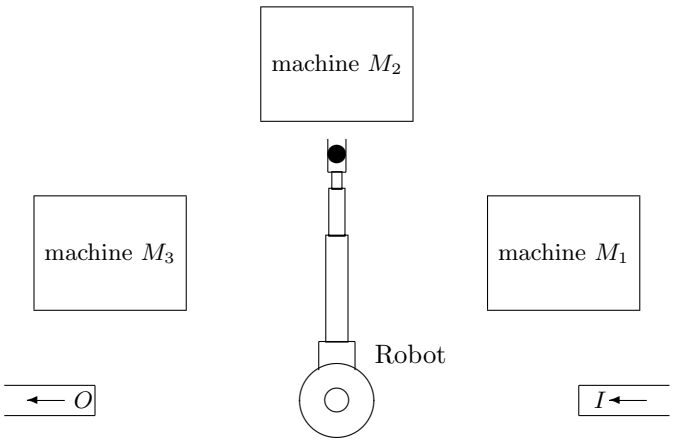

Fig. 1 *A three-machine robotic cell.*

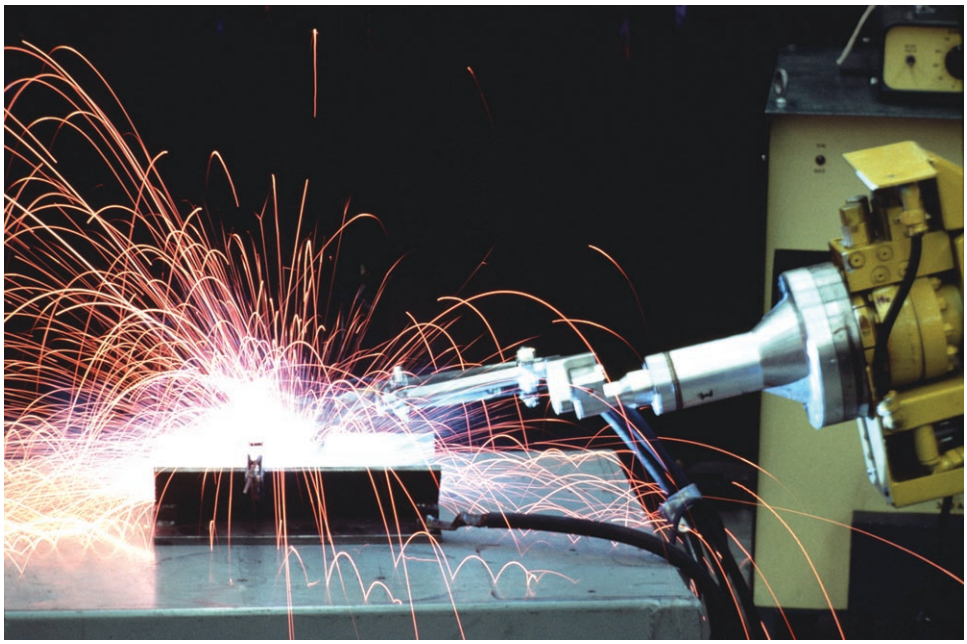

Fig. 2 *A robot with a part at a welding machine.*

is unoccupied. Such cells that have no limitation on the time that a completed part may remain on a machine are common in semiconductor manufacturing [3, 21, 28, 32, 33, 38, 39]. Figure 2 shows the robot with a part at a welding machine which could be one of the machines of the cell shown in Figure 1. Although this study focuses only on these cells, called *free-pickup* cells, the reader should be aware of two other types of cells: in *no-wait* cells, a part must be removed from a machine as soon as that machine has competed its processing. Such conditions are commonly seen in steel manufacturing or plastic molding, where the raw material must maintain a certain temperature, or in food canning to ensure freshness [1, 2, 11, 24, 26, 30]. In *interval processing time* cells, which generalize free-pickup cells and no-wait cells, each stage

has a specific interval of time—a processing time window—for which a part can be processed on that stage. Such cells are commonly found in an electroplating line for printed circuit boards, typically served by an overhead hoist [10, 12, 29].

A standard performance measure used to evaluate the operation of robotic cells is the long-term throughput, which is defined as the average number of finished parts produced per unit of time. A systematic study of the problem of finding optimal sequences of parts and robot moves to maximize the long-term throughput was started by Sethi et al. [35]. Here, we discuss this objective for a robotic cell producing identical parts.

The purpose of this paper is to introduce the readers of this journal to the field of robotic cell scheduling. We provide a theoretical background, a new result concerning the dominance of a certain class of solutions, and descriptions of two fundamental open problems. It is hoped that this will stimulate research in this field by the SIAM community.

2. The Basic Model and Its Variants. We now introduce the basic model of robotic cell scheduling. For a more thorough overview, we refer the reader to surveys by Crama et al. [13] and by Dawande et al. [16].

The m machines of the robotic cell are denoted by $M_1, M_2, \ldots, M_m$. Let $M = \{1, 2, \ldots, m\}$. The input station is denoted by M_0 and the output station by M_{m+1}. We denote the processing time of a part on machine M_i by p_i, $i \in M$. The time required by the robot arm either to load a part onto a machine or to unload a part from a machine M_i is $\epsilon_i \geq 0$. This includes unloading from M_0 (picking a part from the input station) or loading onto M_{i+1} (dropping a part into the output station). The robot's travel time between machines M_i and M_j is δ_{ij}, $0 \leq i, j \leq m+1$. We assume that all data are rational. This is equivalent to assuming that all data are integral: if the data are rational, we may multiply all values by a common denominator to make them integral. Additionally, we assume that all actions and their durations are deterministic, that parts are always available at the input station, and that processing on each machine is non-preemptive.

Three classes of robotic cells, differing in the intermachine travel time and the loading/unloading time, have been considered in the literature.

1. In *additive travel-time robotic cells*, the robot's travel time is $\delta_i \geq 0$ for travel between machines M_i and M_{i+1}, and the total time required to travel between machines M_i and M_j, $i \neq j$, is $\sum_{h=i}^{j-1} \delta_h$, if $i < j$, and $\sum_{h=j}^{i-1} \delta_h$, if $j < i$. The time to load (resp., unload) a part onto (resp., from) machine M_i is $\epsilon_i \geq 0$ (some studies simplify this model by assuming $\delta_i = \delta$ and $\epsilon_i = \epsilon \, \forall i$). Studies which use this travel-time metric include [6, 14, 35].
2. In *constant travel-time robotic cells*, the robot's travel time between any pair of machines M_i and M_j, $i \neq j$, is a constant $\delta \geq 0$. The time to load (resp., unload) a part onto (resp., from) any machine M_i is a constant $\epsilon \geq 0$. Studies which use this travel-time metric include [17, 19].
3. In *Euclidean robotic cells*, the robot travel time from M_i to M_j, $i \neq j$, is $\delta_{ij} \geq 0$, and the travel times satisfy the triangle inequality $\delta_{ij} + \delta_{jk} \geq \delta_{ik} \, \forall i, j, k$. The time to load (resp., unload) a part onto (resp., from) machine M_i is $\epsilon_i \geq 0$. If, in addition, the travel times are *symmetric*, i.e., $\delta_{ij} = \delta_{ji} \, \forall i, j$, we refer to such cells as *Euclidean symmetric robotic cells*. Studies which use this travel-time metric include [1, 9, 29, 30].

The travel time from a machine to itself is zero, i.e., $\delta_{ii} = 0$. The problem data consist of the following: (i) processing time p_i on machine M_i, $i \in M$; (ii) the travel

times from machine M_i to M_j, $i \neq j$; and (iii) the loading and unloading times $\epsilon_i, 0 \leq i \leq m+1$, as defined above.

For a complete description of the movement of parts in a robotic cell, we first need to formalize the transfer of parts, by the robot arm, between successive machines.

DEFINITION. *For each i, $i = 0, \ldots, m$,* activity A_i *consists of the following sequence:*

1. *The robot unloads a part from M_i.*
2. *The robot travels with this part from M_i to M_{i+1}.*
3. *The robot loads this part onto M_{i+1}.*

We will use sequences of activities to specify the movement of parts in the cell.

Example 1. A sequence of four activities $\{A_0, A_2, A_3, A_1\}$ represents the following: the robot picks up a part from M_0, moves to M_1, loads the part onto M_1 (and leaves it to be processed on that machine); moves to M_2 and waits until the part on M_2 has completed processing, unloads the part from M_2, moves to M_3, and loads it onto M_3; waits at M_3 for the entirety of the part's processing, unloads the part from M_3, moves to M_4, and drops it there; moves to M_1, waits until the part on M_1 has been processed, unloads it from M_1, moves to M_2, and loads it onto M_2.

Note that the sequence of activities that describes the robot's loaded moves also completely determines its empty moves.

Since we are considering a bufferless robotic cell, not all sequences of activities are feasible. For example, consider the sequence of activities

$$\pi = \{A_0, A_2, A_3, A_1, A_3, A_2, A_1, A_0\}.$$

Here, after the first execution of activity A_3, machine M_3 does not have a part on it. At the second execution of activity A_3, the robot attempts to unload a nonexistent part from M_3. In general, for an activity sequence to be *feasible* it should not instruct the robot to load a machine which currently contains a part or to unload a machine which does not have a part on it.

For any given feasible sequence of activities $\pi = \langle A_i \rangle$, we define the following.

DEFINITIONS. *A* schedule *is a function $S(A_i, t)$ that assigns a starting time to the tth execution of activity A_i ($i = 0, 1, \ldots, m; t \in \mathbb{N}$). The* long run average throughput*, generally referred to as simply* throughput*, of S is equal to* $\lim_{t\to\infty} \frac{t}{S(A_m,t)}$ *provided the limit exists* [14].

Obtaining a feasible infinite sequence of activities that maximizes throughput is a fundamental problem of robotic cell scheduling. Such a sequence of robotic moves is called *optimal*. Most of the theoretical studies of robotic cell models have focused on a specific class of solutions, namely *cyclic solutions* (or *cyclic schedules*), in which some particular sequence of activities is repeated cyclically. The primary reason for the focus on cyclic schedules is their prevalence in industrial implementations: cyclic schedules are easy to implement, analyze, and control and are therefore preferred by practitioners. However, there is no result in the literature that shows dominance of cyclic schedules, i.e., the sufficiency of considering only cyclic schedules to maximize throughput over all possible schedules (cyclic and noncyclic). We show this dominance result in section 4.

3. Cyclic Schedules. Cyclic production employs a repeatable sequence of activities. For example, $\{A_0, A_2, A_4, A_3, A_1\}$ is a sequence of activities that produces a part in a four-machine cell. Such a sequence can be repeated, with each repetition producing a single part. In general, since a part must be processed on all m machines and then placed into the output station, $m+1$ different activities (exactly one of each

of the $m+1$ activities $A_0, A_1, \ldots, A_m$) are required to produce a part. More precisely, we have the following definition.

DEFINITION. *A* k-unit activity sequence *is a sequence of robot moves which loads and unloads each machine exactly* k *times.*

During cyclic operations, we can restate the definition of feasibility as follows: *for* $i = 1, \ldots, m-1$, *between any two occurrences of* A_i *there must be exactly one* A_{i-1} *and exactly one* A_{i+1}. This condition implies that between any two instances of A_0 there is exactly one A_1, and between any two instances of A_m there is exactly one A_{m-1}. Note that all 1-unit activity sequences are feasible.

An assumption in most studies is that the sequence of robot moves is active.

DEFINITION. *An activity sequence is called* active *if the robot always executes the next operation, whatever that may be, as soon as possible.*

For active sequences, all execution times for the robot's actions (i.e., the schedule $S(A_i, t)$) are uniquely determined once the sequence of activities and the initial state are given. The robot's only possible waiting period can occur at a machine at which the robot has arrived to unload, but the machine has not completed processing its current part. It is known that there is an active sequence which is optimal within the class of 1-unit cycles [37].

DEFINITION. *A* k-unit cycle *is the performance of a feasible* k*-unit activity sequence in a way which leaves the cell in exactly the same state as its state at the beginning of those moves.*

We denote cycles by parentheses, e.g., $(A_0, A_2, A_4, A_3, A_1)$. For present purposes, we can define a robotic cell's *state* by the location of the robot and by which machines currently contain a part. We provide a more rigorous definition of state in section 4. Most studies do not require a rigorous statement of the cell's state because they consider only steady state operations, which we now define.

DEFINITIONS (see [14]). *A robotic cell repeatedly executing a* k*-unit cycle* π *of robot moves is operating in* steady state *if there exist constants* $T(\pi)$ *and* N *such that for every* A_i, $i = 0, \ldots, m$, *and for every* $t \in \mathbb{Z}^+$ *such that* $t > N$, $S(A_i, t+k) - S(A_i, t) = T(\pi)$. *The quantity* $T(\pi)$ *is called the* cycle time *of* π.

It has been shown [7] that, given a feasible initial state, repeating a k-unit activity sequence will enable the robotic cell to reach a steady state (or cyclic solution) in finite time. Therefore, in studies that maximize the long-run average throughput (i.e., assume that the cells operate in steady state for an infinite time), there is no contribution from the initial transient phase. Hence, there is no loss of generality by studying only the steady state behavior.

The *per unit cycle time* of a k-unit cycle π is $T(\pi)/k$. This is the reciprocal of the throughput and is easier to calculate directly. Therefore, rather than maximizing throughput, studies generally focus on minimizing per unit cycle time.

In general, given a k-unit cycle, computing the cycle time is straightforward but not immediate. We list the six possible 1-unit cycles for the 3-machine robotic cell of Figure 1 and illustrate the computation of the cycle time for one of them. Let π_j denote the jth 1-unit cycle, $j = 1, \ldots, 6$: $\pi_1 = \{A_0, A_1, A_2, A_3\}$, $\pi_2 = \{A_0, A_2, A_1, A_3\}$, $\pi_3 = \{A_0, A_1, A_3, A_2\}$, $\pi_4 = \{A_0, A_3, A_1, A_2\}$, $\pi_5 = \{A_0, A_2, A_3, A_1\}$, and $\pi_6 = \{A_0, A_3, A_2, A_1\}$. Note that π_j is defined by a permutation of activities A_i, $i = 1, \ldots, m$.

Example 2. Consider the activity sequence $\pi_5 = \{A_0, A_2, A_3, A_1\}$. We describe the computation of the cycle time $T(\pi_5)$ for constant travel-time cells, i.e., the travel time between any two machines is a constant δ (see Figure 3). The robot picks up a part from M_0 (ϵ), moves to M_1 (δ), loads the part on M_1 (ϵ); moves to M_2 (δ) and

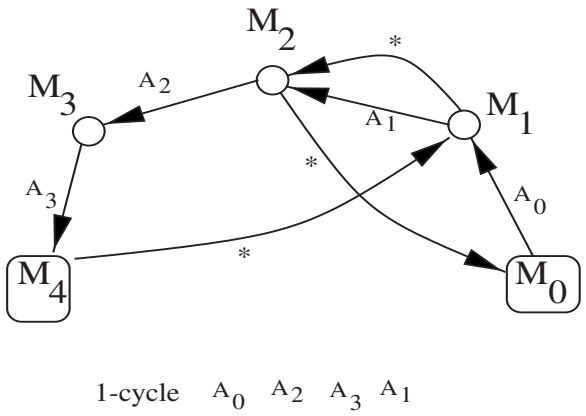

Fig. 3 *Pictorial representation of the* 1*-unit cycle in Example* 2.

waits until the part on M_2 has completed processing (w_2), unloads a part from M_2 (ϵ), moves to M_3 (δ), and loads the part on M_3 (ϵ); waits until the part on M_3 has been processed (p_3), unloads the part from M_3 (ϵ), moves to M_4 (δ), and drops the part at M_4 (ϵ); moves to M_1 (δ), waits until the part on M_1 has been processed (w_1), unloads a part from M_1 (ϵ), moves to M_2 (δ), and loads it onto M_2 (ϵ), then returns to M_0 (δ).

The cycle time expression $T(\pi_5)$ can be derived as follows:

$$\begin{aligned} T(\pi_5) &= \epsilon + \delta + \epsilon + \delta + w_2 + \epsilon + \delta + \epsilon + p_3 + \epsilon + \delta + \epsilon + \delta + w_1 + \epsilon + \delta + \epsilon + \delta \\ &= 7\delta + 8\epsilon + p_3 + w_1 + w_2, \end{aligned}$$

where $w_1 = \max\{0, p_1 - w_2 - p_3 - 4\delta - 4\epsilon\}$ and $w_2 = \max\{0, p_2 - 3\delta - 2\epsilon\}$. By substituting w_1 and w_2 into $T(\pi_5)$, we get $T(\pi_5) = \max\{7\delta + 8\epsilon + p_3, 3\delta + 4\epsilon + p_1, 4\delta + 6\epsilon + p_2 + p_3\}$. The cycle times for the 1-unit cycles π_1, π_2, π_3, π_4, and π_6 can be computed similarly.

Sethi et al. [35] showed that there are exactly $m!$ 1-unit cycles for an m-machine robotic cell. Brauner [4] described an algorithm to generate all k-unit cycles in an m-machine robotic cell. Since the number of distinct k-unit cycles grows rapidly, both with k and m (see, e.g., [8]), obtaining the optimal cyclic solution by explicit enumeration is impractical. That the maximum throughput rate over all cyclic solutions is obtained by repeatedly executing an optimal 1-unit cycle has been proven for additive travel-time cells for $m = 2$ [35] and for $m = 3$ [6, 15]. In fact, for $m = 2$, the optimality of 1-unit cyclic solutions has been shown over all solutions, cyclic and noncyclic [35]. For $m = 4$, Brauner and Finke [5, 7] have constructed problem instances in which the throughput of a 2-unit cycle is better than that of an optimal 1-unit cycle.

Using similar ideas, it can be shown that for constant travel-time cells, repeating an optimal 1-unit cycle provides the best throughput among all cyclic solutions for $m = 2$ and $m = 3$. However, for $m = 4$, there are instances in which the throughput of a 2-unit cycle in such cells is better than that of an optimal 1-unit cycle. We provide an example in section 5.

4. Operating Policies and Dominance of Cyclic Solutions. We now show that under the assumption of rational (or, equivalently, integer) data, it is sufficient to consider the class of cyclic schedules to maximize throughput over all schedules. To

do so, we analyze the operations of a robotic cell as a sequence of states, rather than as a sequence of activities.

To define a general operating sequence for the cell, we first discuss a notion of the state of the cell.

DEFINITION. *The* state *of a robotic cell is specified by the following data:*

- *the occupancy state of each machine, that is, whether a machine contains a part or it is empty;*
- *if a machine contains a part, then the time remaining on its current processing;*
- *the location of the robot;*
- *the occupancy state of the robot, that is, whether the robot arm has a part or not.*

Before we formalize the state space, note that since we are interested in maximizing the throughput of the cell, it is not necessary to consider "wasteful" robot activities such as unnecessary waiting at a location or moving to a location without performing at least one of the loading or unloading operations. Also, since this is a deterministic problem, it is sufficient to define decisions regarding the robot's moves only at those epochs when the robot has just finished loading or unloading a part at a machine. It follows that it is sufficient to consider the state only at these epochs.[1]

Thus, we can represent the state of the cell by the $(m+1)$-tuple $S = (s_1, \ldots, s_{m+1})$, where $s_i \in \{-1, r_i\}, i \in M$. If $s_i = -1$, machine M_i has no part on it; otherwise r_i is the time remaining in the processing of the current part on M_i. $s_{m+1} \in \{A_i, i = 0, \ldots, m\}$ denotes that the robot has just completed activity A_i (i.e., loaded a part onto machine M_{i+1}). For example, if $m = 4$, we could have $S = (5, 0, -1, p_4, A_3)$: the part on M_1 has five time units of processing remaining, M_2 has completed processing a part and that part still resides on M_2, and M_3 is empty. The robot has unloaded a part from M_3, carried it to M_4, and just completed loading it onto M_4.

There is another important observation to be made here. Note that even with integer data, the remaining processing times are in general real numbers. However, since we need to consider the system state only at the epochs mentioned above, the state description will be integral, provided the initial state of the system is restricted to be in integer terms. This restriction can be imposed without loss of generality since some initial adjustments can be made at the beginning to bring the state to integral terms, and the time taken to make these adjustments is of no consequence in the context of the long-term average throughput criterion. Thus, in any state description $S = (s_1, \ldots, s_{m+1})$, $s_i \in \{-1, r_i\}$, we have $r_i \in \{k \in \mathbb{Z} : 0 \leq k \leq p_i\}\, \forall i$. We thus have a *finite-state* dynamic system. Let $\mathcal{S}$ denote the set of all possible state descriptions. For a given cell, we can trivially state the total number of distinct state descriptions in $\mathcal{S}$. Each $s_i, i = 1, \ldots, m$, has $p_i + 2$ possible values and s_{m+1} has $(m + 1)$. Therefore, $(m + 1) \prod_{i=1}^{m} (p_i + 2)$ is the number of all possible state descriptions.

Note that not all state descriptions, as defined above, denote feasible states. For example, for a 4-machine cell, the description $S = (*, -1, -1, *, A_2)$ (where "*" represents any value for the corresponding entry) represents an infeasible state: the robot has just loaded a part onto M_3, so M_3 cannot be empty. To compute a tighter bound on the number of feasible states, we make the following observations:

[1] In the stochastic setting, say, when the processing times are random variables, a throughput maximizing operation may require the robot arm to change its traversal path while the robot is in transition, when new information becomes available. To allow for this, a continuous state space and continuous decision making over time would be required.

(i) for $i = 1, 2, 3, \ldots, m-1$, if $s_{m+1} = A_i$, then $s_{i+1} = p_{i+1}$ and $s_i = -1$;
(ii) if $s_{m+1} = A_0$, then $s_1 = p_1$;
(iii) if $s_{m+1} = A_m$, then $s_m = -1$.

Each state description satisfying the conditions above represents a feasible state. Let $\mathcal{F}$ denote the set of all feasible states. It is easy to see that the total number of feasible states is bounded as follows:

$$|\mathcal{F}| \leq \sum_{i=1}^{m-1} \prod_{\substack{j=1 \\ j \notin \{i, i+1\}}}^{m} (p_j + 2) + \prod_{j=2}^{m} (p_j + 2) + \prod_{j=1}^{m-1} (p_j + 2).$$

We are now ready to define an operating sequence. From here on, we will use the word *state* to mean a feasible state.

DEFINITION. *An* operating sequence *for the cell is an infinite sequence of successive states resulting from feasible operations of the cell starting from an initial state.*

It is important to note that not every infinite sequence of states is feasible. For example, the state $S_1 = (5, 0, -1, p_4, A_3)$ followed by $S_2 = (5, 0, -1, 0, A_4)$ results in an infeasible sequence if $p_4 + \epsilon_4 + \delta_{45} + \epsilon_5 > 0$: since S_2 is the next state of the cell after state S_1, after the robot loads a part onto machine M_4 (state S_1), it waits at that machine for the entire duration while M_4 is processing the part. The robot then unloads the part from M_4 and loads it onto M_5. However, since machine M_1 is busy processing its part during this time, at the instant the robot finishes loading machine M_5 (state S_2), the processing time remaining on M_1 is $\max\{0, 5 - p_4 - \epsilon_4 - \delta_{45} - \epsilon_5\} < 5$.

DEFINITION. *A* policy *for the cell is a function $d : \mathcal{F} \to \mathcal{F}$ such that there exists a state $S \in \mathcal{F}$ for which the infinite sequence $T(d, S) \equiv \{S, d(S), d^2(S), \ldots, d^n(S), \ldots\}$ is an operating sequence.*

LEMMA 1. *For any robotic cell, there exists a throughput maximizing operating sequence which can be generated by a policy.*

Proof. Consider an optimal operating sequence, say, Σ, and suppose that there exists no policy that can generate it. Then, for some state S, the action taken by Σ is different at two (or more) instances when the cell is in state S. Without loss of generality, we can assume that state S occurs in Σ infinitely often, for if the number of occurrences of a state is finite, the segment of Σ up to the last instance of that state can be deleted without affecting its long-term throughput.

Call the average number of finished parts produced per unit of time for the segments of Σ between two successive occurrences of S the *segment-throughput*. If all of the segment-throughputs are equal, we can replace each of these segments by any one segment and maintain the throughput of Σ. Otherwise, replacing a segment having a smaller value of segment-throughput with one having a larger value contradicts the optimality of Σ. Thus, there exists a throughput maximizing operating sequence which can be generated by a policy. □

Given that the cell is currently in state $S \in \mathcal{F}$, the functional image $d(S)$ of a policy d completely specifies the transition to the next state, and thus completely defines the robot's action. Together, a policy d and an initial state $S_0 \in \mathcal{F}$ generate a unique operating sequence $\{S_0, d(S_0), d^2(S_0), \ldots, d^n(S_0), \ldots\}$. We would like to emphasize that an initial state is required to specify an operating sequence generated by a policy. To illustrate, suppose $\mathcal{F} = \{S_1, S_2, \ldots, S_6\}$ and d is defined as follows: $d(S_i) = S_{i+1}, i = 1, 2, 4, 5; d(S_3) = S_1, d(S_6) = S_4$. If the initial state is S_1, we obtain the sequence $\{S_1, S_2, S_3, S_1, S_2, S_3, \ldots\}$. If the initial state is S_4, we obtain $\{S_4, S_5, S_6, S_4, S_5, S_6, \ldots\}$.

Let $\rho(d, S)$ be the throughput of the operating sequence $T(d, S)$. The maximum throughput, $\rho(d)$, obtainable from a policy d is then $\max_{S\in\mathcal{F}}\{\rho(d, S) : T(d, S)$ is an operating sequence$\}$. Note that the maximum exists since $|\mathcal{F}|$ is finite. The maximum throughput of the cell is obtained by maximizing $\rho(d)$ over all policies d. Since a policy is a function with domain and range on the finite set $\mathcal{F}$, the total number of distinct policies is at most $|\mathcal{F}|^{|\mathcal{F}|}$. Moreover, since an operating sequence is completely specified by a policy and an initial state, the total number of operating sequences is at most $|\mathcal{F}|^{(|\mathcal{F}|+1)}$. We record these observations below.

LEMMA 2. *The total number of policies is at most* $|\mathcal{F}|^{|\mathcal{F}|}$. *The total number of operating sequences is at most* $|\mathcal{F}|^{(|\mathcal{F}|+1)}$.

The finiteness of $\mathcal{F}$ implies that the infinite sequence of states resulting from *any* policy is a repeating sequence. Consequently, the sequence of robot actions is a repeating sequence. Every policy repeats a minimal sequence of robot moves. The minimal sequence is a state-preserving sequence: the state of the cell at the beginning is identical to the state of the cell at the end of the sequence. We therefore refer to a sequence resulting from a policy as a *cyclic sequence*. The discussion above and Lemma 1 yield the following result.

THEOREM 3. *There exists a cyclic sequence of robot moves which maximizes long-term throughput of the robotic cell.*

It is therefore sufficient to optimize over the class of cyclic sequences. This result provides a sound basis for the widely used industry practice of specifying the operation of a robotic cell via cyclic sequences.

5. Fundamental Open Problems. In the previous section, we derived an upper bound on the cardinality of the set of feasible states $\mathcal{F}$. Note that, by definition, a k-unit cycle has $k(m+1)$ distinct states (or $k(m+1)$ activities in the terminology of section 3). It follows that an upper bound on the number of parts produced by any throughput maximizing cycle is

$$\eta = \frac{|\mathcal{F}|}{(m+1)}.$$

Let $\mathcal{C}_k, k = 1, \ldots, \eta$, denote the class of all k-unit cycles. Then the k-unit cycle C_k^* with $T(C_k^*) = \min_{C_k\in\mathcal{C}_k} T(C_k)$ can be executed indefinitely to obtain a k-unit cyclic solution with maximum throughput. A cyclic solution with cycle time $T(C^*) = \min_{k=1,\ldots,\eta} T(C_k^*)$ then maximizes the throughput over all cyclic solutions.

Thus, from an algorithmic point of view, the following problems become relevant:

1. *Problems k-OPT.* For additive (resp., constant, Euclidean) travel-time cells, provide algorithms to find an optimal k-unit cycle for a given k, $1 \le k \le \eta$.
2. *Problem OPT.* For additive (resp., constant, Euclidean) travel-time cells, provide an algorithm to find an optimal k-unit cycle over all k, $1 \le k \le \eta$.

First, we would like to emphasize the need for efficient and practicable algorithms for solving these problems. Since the number of cyclic solutions is finite, a complete enumeration of such solutions (or any other method that requires enumerating a prohibitively large search space) will provide a valid answer to these problems; however, such methods do not provide a satisfactory resolution as they cannot be implemented in practice within reasonable time. In terms of computational complexity theory [18], we either want to obtain a polynomial-time algorithm for a problem or show that it is NP-hard. While demonstrating that a problem is NP-hard does not completely rule out practicable solution methods for solving it to optimality, the use of heuristics to obtain approximate solutions becomes much more acceptable in such a case.

Second, note that the most significant among these problems is Problem OPT, the problem of obtaining a cyclic schedule with maximum throughput. While solving Problems k-OPT, $k = 1, 2, \ldots, \eta$, automatically solves Problem OPT, it may not be necessary to do so. In fact, as we mention below, for all the special cases in which problem OPT has been solved, the result follows without analyzing the individual problems k-OPT, $k = 1, 2, \ldots, \eta$. Over the past decade, a significant amount of research in the scheduling of robotic cells has, directly or indirectly, focused on answering these two problems. We now discuss the results available for these problems and our attempts at solving the unresolved questions.

For the three classes of robotic cells discussed in section 1, Problem k-OPT has been resolved for $k = 1$. For additive travel-time cells, Crama and van de Klundert [14] used dynamic programming to obtain a polynomial-time algorithm for an optimal 1-unit cycle. For constant travel-time cells, Dawande et al. [17] proposed a polynomial-time algorithm that repeatedly uses a solution of the shortest path problem in directed acyclic graphs within a binary search procedure. Brauner, Finke, and Kubiak [9] showed that the problem of finding an optimal 1-unit cycle in Euclidean travel-time cells is NP-hard in the strong sense.

No algorithmic results are available for Problems k-OPT, $k \geq 2$. However, we do know that these problems are relevant since examples showing that the throughput of an optimal 2-unit cycle is strictly better than the throughput of the optimal 1-unit cycle have been demonstrated. In other words, there exist instances of cells in which $T(C_2^*) < T(C_1^*)$. For additive travel-time cells, Brauner and Finke [5, 7] showed such instances. For constant travel-time cells, since no examples have been published, we provide the following.

Example 3. Consider a 4-machine robotic cell where the processing times of machines $M_i, i = 1, \ldots, 4$, are 21, 3, 1, and 23, respectively. Let $\delta = 4$ and $\epsilon = 0$. The optimal 2-unit cycle is

$$C_2^* = (A_0, A_4, A_3, A_1, A_0, A_4, A_2, A_3, A_1, A_2)$$

with a per unit cycle time of 39.5, while an optimal 1-unit cycle is $C_1^* = (A_0, A_4, A_3, A_2, A_1)$ with a cycle time of 40. The optimality of C_1^* and C_2^* is verified by evaluating the cycle times of all 1-unit and 2-unit cycles, respectively.

For Problem OPT, two different types of results are available: (i) those that solve the problem under specific conditions on the problem data, and (ii) those that provide a bound on the optimal throughput in terms of the throughput of an optimal 1-unit cycle.

As discussed in section 2, the problem data consists of the number of machines and their processing times, the intermachine travel times for the robot, and the time required to load (resp., unload) a part onto (resp., from) a machine. Under different conditions on the problem data, several results use the following two-step procedure to solve Problem OPT.

1. Obtain an upper bound on the maximum throughput over all cyclic solutions. For a k-unit cycle C_k, this is typically done by estimating a lower bound on the per-unit cycle time, $\frac{T(C_k)}{k}$. Here, the analysis uses quantities such as the minimum number of robot moves involved, the number of loadings and unloadings, and the minimum processing time required.
2. Provide a specific cyclic solution that achieves the upper bound (derived in step 1 above) on the throughput, thus establishing its optimality.

For an example of a 1-unit cycle that is known to be optimal over all k-unit cycles under certain conditions, consider the reverse cycle in an m-machine cell: $\pi_D = (A_0, A_m, A_{m-1}, \ldots, A_1)$. To perform π_D, the robot unloads a part from the input station (M_0), carries it to M_1, and loads M_1. It then travels to M_m, unloads M_m, and carries that part to the output station (M_{m+1}). It repeats this sequence for $i = m-1, m-2, \ldots, 1$: travel to M_i, unload M_i, carry the part to M_{i+1}, load M_{i+1}. After loading M_2 (which completes activity A_1), the robot completes the cycle by traveling to the input station (M_0). At each machine, before unloading a part from it, the robot may have to wait for that machine to complete processing.

The following results detail the circumstances under which repeating the 1-unit cycle indefinitely is an optimal cyclic solution.

THEOREM 4 (see [16]). *Cycle π_D is optimal in a Euclidean travel-time cell if*

$$\max_{1\le i\le m}\{p_i + \delta_{i,i+1} + \delta_{i+1,i-1} + \delta_{i-1,i} + 4\epsilon\} \ge 2(m+1)\epsilon + \sum_{i=0}^{m}\delta_{i,i+1} + \sum_{i=2}^{m+1}\delta_{i,i-2} + \delta_{1,m}.$$

COROLLARY 5 (see [17]). *Cycle π_D is optimal in a constant travel-time cell if* $\max_{1\le i\le m} p_i + 3\delta + 4\epsilon \ge 2(m+1)(\delta+\epsilon)$.

COROLLARY 6 (see [14]). *Cycle π_D is optimal in an additive travel-time cell in which $\delta_i = \delta$ and $\epsilon_i = \epsilon$ $\forall i$ if* $\max_{1\le i\le m} p_i + 4\delta + 4\epsilon \ge 4m\delta + 2(m+1)\epsilon$.

THEOREM 7 (see [17]). *π_D is optimal in a constant travel-time cell if $p_i \ge \delta$ $\forall i$.*

THEOREM 8 (see [16]). *π_D is optimal in an additive travel-time cell in which $\delta_i = \delta$ and $\epsilon_i = \epsilon$ $\forall i$ if*

$$p_i + p_{i+1} \ge (4m-6)\delta + 2(m-2)\epsilon,\ i = 1, \ldots, m-1.$$

The second type of result for Problem OPT estimates the gap between the throughputs of the optimal cyclic solution and the optimal 1-unit cyclic solution. There are at least two reasons that motivate the estimation of this gap. (i) On the one hand, if the gap is very small, it can be used to devise new search methods that start with the optimal 1-unit solution and seek to improve its throughput by local search. Here, the hope is that the small gap can be closed by a structured, limited search and thus result in an efficient algorithm. On the other hand, a large gap indicates that fundamentally new methods (from those employed for finding an optimal 1-unit cycle) are likely to be required to obtain an optimal cyclic solution. (ii) Additionally, the gap is useful to assess the benefit of the optimal cyclic solution over that of an optimal 1-unit cyclic solution. This can be significant insight, especially in practice. For additive cells, Crama and van de Klundert [14] showed that the throughput of an optimal 1-unit cycle is at least $\frac{1}{2}$ that of an optimal cyclic solution. Geismar, Dawande, and Sriskandarajah [20] improved this ratio to $\frac{2}{3}$. For constant (resp., Euclidean) cells, the best known ratio is $\frac{2}{3}$ (resp., $\frac{1}{4}$) [20]. Clearly, these ratios are too weak to pursue the local search strategy mentioned in (i) above. However, it is important to note that none of these ratios has been shown to be tight; i.e., for none of the classes of cells has it been shown that the above bounds are the largest possible for the ratio of the throughput of an optimal 1-unit cycle to the throughput of an optimal cyclic solution. Thus, the precise estimation of the gap remains open for the various classes of cells.

Another direction that we have attempted to explore is that of using mathematical programming formulations. Here, the idea is to express the problem of maximizing

throughput of a cyclic solution as either a linear program or an efficient integer linear program (e.g., one in which the constraint matrix is totally unimodular [31]). For a given k-unit cyclic solution, the computation of its cycle time (and, hence, its throughput) can indeed be expressed as a linear program [21, 28]. However, our attempts to formulate the problem of searching over all k-unit cyclic solutions have been unsuccessful so far.

We conclude this paper by mentioning two other problems that have not been addressed in the literature but might be helpful in understanding the structure of cyclic solutions in robotic cells.

It is not known whether the maximum throughput of a cell over the class of k-unit cycles is an increasing function of k, $k \geq 1$. For example, it is not known if the maximum throughput over all 3-unit cycles is at least as large as that over all 2-unit cycles. Indeed, nothing is known about the behavior of the maximum throughput with respect to k. However, if we define Problem $^{\leq}k$-OPT as that of determining a throughput maximizing cyclic solution over all l-unit cycles, $1 \leq l \leq k$, then it follows immediately that the throughput of an optimal solution of $^{\leq}k$-OPT is a weakly increasing function of k. However, there are no results on the rate of change of this increase. For instance, if the increase is decreasing in k (i.e., diminishing marginal increase), it might be reasonable to expect that Problem OPT attains its optimal solution at a relatively small (as compared to η) value of k.

The dominance of cyclic solutions and most of the algorithmic results for robotic cells assume that all cell data are rational. This is a reasonable assumption in practice and greatly reduces the state space of the cell. Results for arbitrary real data, while not necessary for practical applications, seem to be much more challenging mathematically, and none are available in the literature.

REFERENCES

[1] A. Agnetis, *Scheduling no-wait robotic cells with two and three machines*, European J. Oper. Res., 123 (2000), pp. 303–314.

[2] A. Agnetis and D. Pacciarelli, *Part sequencing in three machine no-wait robotic cells*, Oper. Res. Lett., 27 (2000), pp. 185–192.

[3] E. Akçali, K. Nemoto, and R. Uzsoy, *Cycle-time improvements for photolithography process in semiconductor manufacturing*, IEEE Trans. Semiconductor Manufacturing, 14 (2001), pp. 48–56.

[4] N. Brauner, *Ordonnancement dans des cellules robotisées*, *Thèse de doctorat*, Université Joseph Fourier, Grenoble, France, 1999.

[5] N. Brauner and G. Finke, *Final Results on the One-Cycle Conjecture in Robotic Cells*, Internal note, Laboratoire LEIBNIZ, Institut IMAG, Grenoble, France, 1997.

[6] N. Brauner and G. Finke, *On the conjecture in robotic cells: New simplified proof for the three-machine case*, INFOR, 37 (1999), pp. 20–36.

[7] N. Brauner and G. Finke, *Cycles and permutations in robotic cells*, Math. Comput. Modelling, 34 (2001), pp. 565–591.

[8] N. Brauner and G. Finke, *Optimal moves of the material handling system in a robotic cell*, Internat. J. Production Economics, 74 (2001), pp. 269–277.

[9] N. Brauner, G. Finke, and W. Kubiak, *Complexity of one-cycle robotic flow-shops*, J. Scheduling, 6 (2003), pp. 355–372.

[10] A. Che, C. Chu, and F. Chu, *Multicyclic hoist scheduling with constant processing times*, IEEE Trans. Robotics and Automation, 18 (2002), pp. 69–80.

[11] A. Che, C. Chu, and E. Levner, *A polynomial algorithm for* 2*-degree cyclic robot scheduling*, European J. Oper. Res., 145 (2003), pp. 31–44.

[12] H. Chen, C. Chu, and J. Proth, *Cyclic scheduling with time window constraints*, IEEE Trans. Robotics and Automation, 14 (1998), pp. 144–152.

[13] Y. Crama, V. Kats, J. van de Klundert, and E. Levner, *Cyclic scheduling in robotic flowshops*, Ann. Oper. Res., 96 (2000), pp. 97–124.

[14] Y. Crama and J. van de Klundert, *Cyclic scheduling of identical parts in a robotic cell*, Oper. Res., 45 (1997), pp. 952–965.
[15] Y. Crama and J. van de Klundert, *Cyclic scheduling in 3-machine robotic flow shops*, J. Scheduling, 2 (1999), pp. 35–54.
[16] M. Dawande, H. N. Geismar, S. Sethi, and C. Sriskandarajah, *Sequencing and scheduling in robotics cells: Recent developments*, J. Scheduling, 8 (2005), pp. 387–426.
[17] M. Dawande, C. Sriskandarajah, and S. Sethi, *On throughput maximization in constant travel-time robotic cells*, Manufacturing and Service Operations Management, 4 (2002), pp. 296–312.
[18] M. Garey and D. Johnson, *Computers and Intractability: A Guide to NP-Completeness*, W. H. Freeman, New York, 1979.
[19] H. N. Geismar, M. Dawande, and C. Sriskandarajah, *Robotic cells with parallel machines: Throughput maximization in constant travel-time cells*, J. Scheduling, 7 (2004), pp. 375–395.
[20] H. N. Geismar, M. Dawande, and C. Sriskandarajah, *Approximation algorithms for k-unit cyclic solutions in robotic cells*, European J. Oper. Res., 162 (2005), pp. 291–309.
[21] H. N. Geismar, N. Ramanan, and C. Sriskandarajah, *Increasing throughput for robotic cells with parallel machines and multiple robots*, IEEE Trans. Automat. Sci. Engrg., 1 (2004), pp. 84–89.
[22] N. Hall, H. Kamoun, and C. Sriskandarajah, *Scheduling in robotic cells: Classification, two and three machine cells*, Oper. Res., 45 (1997), pp. 421–439.
[23] N. Hall, H. Kamoun, and C. Sriskandarajah, *Scheduling in robotic cells: Complexity and steady state analysis*, European J. Oper. Res., 109 (1998), pp. 43–63.
[24] N. Hall and C. Sriskandarajah, *A survey of machine scheduling problems with blocking and no-wait in process*, Oper. Res., 44 (1996), pp. 510–525.
[25] H. Kamoun, N. Hall, and C. Sriskandarajah, *Scheduling in robotic cells: Heuristics and cell design*, Oper. Res., 47 (1999), pp. 821–835.
[26] V. Kats and E. Levner, *Cyclic scheduling in a robotic production line*, J. Scheduling, 5 (2002), pp. 23–41.
[27] V. Kats and Z. Mikhailetsky, *Exact solution of a cyclic scheduling problem*, Autom. Remote Control, 4 (1990), pp. 187–190 (in Russian).
[28] S. Kumar, *Analytical and Metaheuristic Solutions for Emerging Scheduling Problems in E-commerce and Robotics*, Ph.D. thesis, The University of Texas at Dallas, 2001.
[29] L. Lei and T. Wang, *Determining optimal cyclic hoist schedules in a single-hoist electroplating line*, IIE Trans., 26 (1994), pp. 25–33.
[30] E. Levner, V. Kats, and V. Levit, *An improved algorithm for cyclic robotic flowshop scheduling in a robotic cell*, European J. Oper. Res., 97 (1997), pp. 500–508.
[31] G. Nemhauser and L. Wolsey, *Integer Programming and Combinatorial Optimization*, Wiley, New York, 1988.
[32] T. Perkinson, P. McLarty, R. Gyurcsik, and R. Cavin, *Single-wafer cluster tool performance: An analysis of throughput*, IEEE Trans. Semiconductor Manufacturing, 7 (1994), pp. 369–373.
[33] T. Perkinson, R. Gyurcsik, and P. McLarty, *Single-wafer cluster tool performance: An analysis of the effects of redundant chambers and revisitation sequences on throughput*, IEEE Trans. Semiconductor Manufacturing, 9 (1996), pp. 384–400.
[34] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1995.
[35] S. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak, *Sequencing of parts and robot moves in a robotic cell*, Internat. J. Flexible Manufacturing Systems, 4 (1992), pp. 331–358.
[36] C. Sriskandarajah, N. Hall, and H. Kamoun, *Scheduling large robotic cells without buffers*, Ann. Oper. Res., 76 (1998), pp. 287–321.
[37] J. van de Klundert, *Scheduling Problems in Automated Manufacturing*, Ph.D. thesis, Maastricht University, The Netherlands, 1996.
[38] S. Venkatesh, R. Davenport, P. Foxhoven, and J. Nulman, *A steady-state throughput analysis of cluster tools: Dual-blade versus single-blade robots*, IEEE Trans. Semiconductor Manufacturing, 10 (1997), pp. 418–424.
[39] S. Wood, *Simple performance models for integrated processing tools*, IEEE Trans. Semiconductor Manufacturing, 9 (1996), pp. 320–328.