Cyclic scheduling of identical parts in a robotic cell

Yves Crama [1] [2]
Joris van de Klundert [2]

February 17, 1995

[1]Département de Gestion, Université de Liège, 4000 Liège, Belgium
[2]Department of Quantitative Economics, Faculty of Economics, University of Limburg, Maastricht, The Netherlands

Abstract

We consider a robotic flowshop in which one type of product is to be repeatedly produced, and where transportation of the parts between the machines is performed by the robot. The identical parts cyclic scheduling problem is then to find a shortest cyclic schedule for the robot, i.e. a sequence of robot moves that can be repeated infinitely many times and has minimum cycle time. This problem has been solved by Sethi et al. [1992] when $m \leq 3$. In this paper, we considerably generalize their results by proving that the identical parts cyclic scheduling problem can be solved in time polynomial in m, where m denotes the number of machines in the shop. In particular, we present a dynamic programming approach that allows to solve the problem in $O(m^3)$ time. Our analysis heavily relies on the concept of pyramidal permutation, a concept previously investigated in connection with the traveling salesman problem.

Keywords: Manufacturing, automated systems : materials handling in robotic cells. Production/Scheduling, sequencing : flow shop, cycle time minimization. Dynamic programming, deterministic : traveling salesman, pyramidal permutations.

1 Introduction

Recently, scheduling problems arising in flexible manufacturing cells, flexible flowlines and similar automated production systems have received much attention in the literature. In such environments, transportation of the parts between the machines is usually performed by an automated material handling system, be it a conveyor, or a pool of automatically guided vehicles (AGVs), or a robot. Much of the scheduling literature, however, has ignored the constraints placed by material handling devices on the efficiency of the productive system, either because these devices were not regarded as bottlenecks, or, more pragmatically, for reasons of modeling simplicity. Only recently has material handling been paid special attention and been incorporated explicitly in scheduling models (see e.g. Blazewicz et al. [1991], Hall et al. [1993a, 1993b], Hall et al. [1994], Jeng et al. [1993], King et al. [1992], Kise [1991], Kise et al. [1991], Krishnamurthy et al. [1993], Sethi et al. [1992]).

In this paper, we investigate a cyclic scheduling problem for a robotic flowshop whose throughput rate is highly dependent on the interaction between the material handling system (namely, the robot) and the machines. More precisely, we consider a robotic flowshop consisting of m machines, an input device, an output device and a robot (see Figures 1 and 2). There are no buffers in the flowshop (a similar problem with buffers is considered in King et al. [1993]). Transportation of the parts between the machines is taken care of by the robot, which can only handle one part at a time. In the most general setting of the problem, a so-called minimal part set (MPS) is to be repeatedly produced, where the MPS consists of parts of different types in proportion to a certain target production mix (see e.g. Stecke [1983]). The objective of the scheduling problem is then to determine the part input sequence (i.e., the order in which the parts in the MPS should be processed) and the corresponding sequence of robot moves so as to maximize the long run throughput rate or to minimize the long run cycle time of the system.

This problem (and closely related ones) has been considered by several authors (Sethi et al. [1992] and Hall et al. [1993a] provide references). Sethi et al. [1992] showed that, when there are only two machines (and under some restrictions on the move sequences that the robot is allowed to perform), the problem can be solved in polynomial time. The same result was obtained by Kise et al. [1991] for a makespan minimization objective. On the other hand, Hall et al. [1993b] proved that the problem is already strongly NP-hard for a three-machine robotic flowshop. As a matter of fact, these authors established that computing the optimal part input sequence in a three-machine flowshop is strongly NP-hard when certain robot move sequences are treated as given.

In our work, by contrast, we restrict ourselves to the special case of the problem where the number of machines is arbitrary, but all parts are of the same type. In this framework, the part input sequencing problem vanishes altogether and the term *cyclical*, that usually indicates in the literature that the part input sequence repeats identically for each and every MPS (see e.g. Agnetis et al. [1993], Karabati & Kouvelis [1993], McCormick et al. [1989]), applies here only to the sequence of moves performed by the robot.

The resulting *identical parts cyclic scheduling problem* has been investigated by Sethi et al. [1992] and Hall et al. [1993a]. More precisely, in the classification scheme of Hall et al. [1993a], we are interested in the problem RCC$m|k = 1$,1-unit$|C_t$, meaning that the robotic cell contains m machines, that there is exactly one part type, and that the objective is to minimize the cycle time C_t under the restriction that one unit be produced in each cycle. In particular, Sethi et al. [1992] described a simple decision rule that computes the optimal robot move sequence

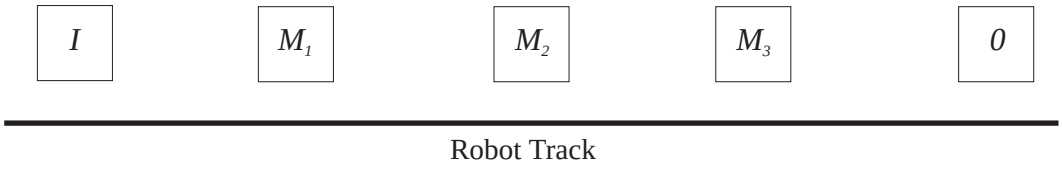

Figure 1: A 3-machine robotic cell (line layout)

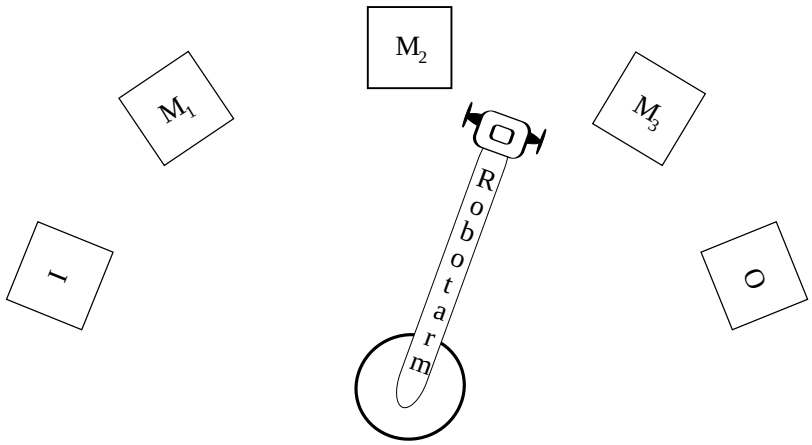

Figure 2: A 3-machine robotic cell (circle layout)

when there are only three machines in the flowshop. In this paper, we considerably extend their analysis by proving that the identical parts cyclic scheduling problem can be solved in time polynomial in m, where m denotes the number of machines in the shop.

In Section 2, we give a more precise definition of the identical parts cyclic scheduling problem, and we describe a one-to-one correspondence (discovered by Sethi et al. [1992]) between its feasible solutions and the permutations of the set $\{1, \ldots, m\}$. In Section 3, we derive upper and lower bounds on the optimal cycle time. We also present in this section the key result of our paper, namely that the set of pyramidal permutations necessarily contains an optimal solution of the problem (pyramidal permutations have been previously introduced in the framework of the traveling salesman problem; see e.g. Gilmore et al. [1985]). In Section 4, we give an efficient algorithm to compute the cycle time of a schedule described by a pyramidal permutation. Relying on this result, we present in Section 5 a dynamic programming approach that allows to solve the recognition version of the identical parts cyclic scheduling problem in $O(m^2)$ time, and its optimization version in $O(m^3)$ time. Finally, we discuss in Section 6 some directions for further research.

2 Cycles, Permutations & Schedules

In this section we discuss the input parameters of the problem and its objective. A solution for the problem is defined as a sequence of robot moves that maximizes the long run throughput rate. The problem is shown to be a permutation problem. Furthermore, the objective of the problem is restated in terms of schedules and cycle times, rather than throughput rates.

Let us first define the notation we use for the entities that play a role in the problem. The m machines of the robotic cell are denoted by $M_1 \dots M_m$. The input device is denoted by I or M_0. The output device is denoted by O or M_{m+1}. Each part is initially available at the input device and must be processed succesively by $M_1, M_2, \dots, M_m$, until it is unloaded at the output device. Each machine can only process one part at a time and there are no buffers for intermediary storage at the machines. We denote the processing time of the part on machine M_i by p_i, $i = 1 \dots m$. We call the segment of the robot track between two adjacent machines a trajectory, and we denote by δ_i the time the robot needs to travel from machine M_i to M_{i+1} , or from M_{i+1} to M_i, $i = 0, \dots, m$. Loading a part onto $M_i, i = 1, \dots, m+1$ or unloading a part from $M_i, i = 0, \dots, m$ takes time ϵ_i. Hence the input of the problem consists of:

- processing times $p_1, \dots, p_m$
- travel times $\delta_0, \dots, \delta_m$
- (un)loading times $\epsilon_0, \dots, \epsilon_{m+1}$

For reasons of clarity we usually assume $\delta_i = \delta, i = 0, \dots, m$, $\epsilon_i = \epsilon, i = 0, \dots, m+1$. However, all results presented go through for trajectory and machine dependent travel and (un)loading times.

Let us now describe the type of robot moves that we want to consider. From a practical viewpoint it is not desirable to specify all moves the robot has to perform until a complete batch is processed, since the batch size may be fairly large (we assume it to be infinite). Hence we will be interested in more compact sequences that the robot can execute a number of times. More precisely, we will be interested in sequences with the property that exactly one part is taken from the input device (and one part is dropped at the output device) in each execution of the sequence. Such sequences of robot moves are called 1-unit cycles :

Definition 2.1. A *1-unit cycle* is a sequence of robot moves in which each machine is loaded and unloaded exactly once.

Observe that a 1-unit cycle returns the cell in its original state and hence can be repeated infinitely many times. Sethi et al. [1992] conjecture that the maximum throughput rate which can be achieved executing a 1-unit cycle equals the maximum throughput rate over all sequences of robot moves. A weak form of this conjecture has been proved by Hall et al. [1992] for the identical parts 3-machine cyclic scheduling problem. The conjecture provides further motivation for restricting our attention to 1-unit cycles.

Sethi et al. [1992] have the following theorem on the number of possible 1-unit cycles in a robotic cell with m-machines:

Theorem 2.1. (Sethi et al.) In a robotic cell with m machines, there are exactly $m!$ 1-unit cycles.

The following definition is helpful to understand Theorem 2.1.

Definition 2.2. For all $i, i = 0 \ldots, m$, *activity* A_i consists of the following sequence of robot moves :

1. unload M_i
2. travel from M_i to M_{i+1}
3. load M_{i+1}

Without loss of generality, it may be assumed that every 1-unit cycle starts with the robot moves as specified by A_0. The proof of Theorem 2.1 establishes that every 1-unit cycle defines a permutation of the activities starting with A_0 and, conversely, that every permutation of the activities starting with A_0 corresponds to a 1-unit cycle. Thus, computing an optimal 1-unit cycle is equivalent to computing an optimal permutation of the activities. In the sequel, we will use the names "1-unit cycle" and "permutation of the activities" interchangeably.

Let us now concentrate on the objective function of our problem. Informally speaking, we want to maximize the long run average throughput rate of the system, or equivalently, we want to minimize its long run average cycle time. To make this concept more precise, consider the following definitions.

Definition 2.3. A *schedule* is a function $S(A_i, t)$ that assigns a starting time to the t-th execution of activity A_i ($i = 0, \ldots, m,\ t \in \mathbb{N}$). The *long run average cycle time* of S is equal to

$$\lim_{t \to \infty} \frac{S(A_m, t)}{t}$$

assuming that the limit exists.

Definition 2.4. A schedule S is called a *steady state schedule* if there exists a constant L (called the *cycle time* of S) such that for every $A_i, i = \ 0, \ldots m$, and for every $t \in \mathbb{N}, S(A_i, t+1) - S(A_i, t) = L$.

Definition 2.5. Given a permutation of the activities, say $\pi = (A_{i_0}, A_{i_1}, \ldots, A_{i_m})$, and a schedule $S(A_i, t)$, we say that S is *a schedule for* π if the sequence of activities defined by S is consistent with π, i.e. $S(A_{i_j}, t) < S(A_{i_k}, t)$ for all $j, k \in \{0, \ldots, m\}$ with $j < k$ and for all $t \in \mathbb{N}$.

Clearly, for a steady state schedule, the long run average cycle time coincides with the cycle time. Van de Klundert [1994] proves that, for each 1-unit cycle, there exists a steady state schedule S that minimizes the long run average cycle time over all schedules. (This conclusion could also be drawn from an analysis of the periodical behavior of the cell, viewed as a discrete

system; see e.g. Cohen et al [1985], Sethi et al. [1992].

Definition 2.6. Let π be a permutation of the activities. The *cycle time* of π, denoted $L(\pi)$, is the minimum cycle time achievable by a steady state schedule for π.

With these definitions at hand, we can formulate as follows the *identical parts cyclic scheduling problem* : given processing times $p_1, \ldots, p_m$, travel times $\delta_0, \ldots, \delta_m$ and (un)loading times $\epsilon_0, \ldots, \epsilon_{m+1}$, find a permutation of the activities with minimum cycle time.

3 Pyramidal Permutations.

In this section, we first give a lower bound on the cycle time of the optimal permutation, and we describe a permutation whose cycle time never exceeds twice the lower bound. These results and their derivation may help the reader gain some intuition for the problem, and will also play a role in the analysis presented in Sections 4 and 5. In the second part of the section, we introduce pyramidal permutations and show that the set of pyramidal permutations necessarily contains an optimal 1-unit cycle.

Lemma 3.1. The cycle time $L(\pi)$ of every permutation π satisfies:

$$L(\pi) \geq \max\{2(m+1)(\delta+\epsilon), \max_i p_i + 4(\delta+\epsilon)\}$$

Proof. Consider a permutation π of the activities and assume without loss of generality that π starts with A_0. Since the next cycle starts again with A_0, in any cycle the robot must at least travel from I to O and back to I, which induces a travel time of at least $2\delta(m+1)$. Also, in any cycle every machine must be loaded and unloaded, the input must be unloaded and the output must be loaded; hence the total time the robot spends loading and unloading machines is at least $2\epsilon(m+1)$. Thus we have that $L(\pi) \geq 2(m+1)(\delta+\epsilon)$.

To prove that $L(\pi) \geq \max_i p_i + 4(\delta+\epsilon)$, fix $i \in \{1, \ldots, m\}$, and consider an optimal steady state schedule for π, say S. Then, $L(\pi) = S(A_i, t+1) - S(A_i, t)$, i.e. the cycle time equals the time between two consecutive unloading operations of machine M_i. Now consider the point in time τ between $S(A_i, t)$ and $S(A_i, t+1)$ at which M_i starts processing. Between $S(A_i, t)$ and τ, the robot must at least have performed A_i and A_{i-1}. Hence we have $\tau \geq S(A_i, t) + 4\delta + 4\epsilon$. Furthermore, the unloading operation starting at $S(A_i, t+1)$ cannot be performed before machine M_i has finished processing the part, i.e. $S(A_i, t+1) \geq \tau + p_i$. From these two inequalities we deduce $L(\pi) \geq p_i + 4\delta + 4\epsilon$, which concludes the proof. ∎

If the robot is relatively slow, its travel time is likely to be the bottleneck of the system. In this case, the permutation $A_0, A_1, \ldots, A_m$, to be called π_U, might well be the optimal permutation since it has minimum travel time. On the other hand, if the robot is relatively fast, the permutation $A_0, A_m, A_{m-1}, \ldots, A_1$, to be called π_D, appears to be a good alternative, since it allows each machine as much time for processing as possible. We now derive an expression for $L(\pi_D)$:

Lemma 3.2.

$$L(\pi_D) = \max\{4m\delta + 2(m+1)\epsilon, \max_i p_i + 4(\delta + \epsilon)\}$$

Proof. The total travel time, and load/unload time for π_D is equal to $4m\delta + 2(m+1)\epsilon$ and is a lowerbound for $L(\pi_D)$. By Lemma 3.1 we know that $L(\pi_D) \geq \max_i p_i + 4\delta + 4\epsilon$. Thus the maximum over these two is a lowerbound for $L(\pi_D)$. Let C equal this maximum. We give a schedule for π_D with cycle time C and prove its feasibility by induction. Observe that a schedule is feasible if the robot can indeed reach every machine in time, and never unloads a machine before it has finished processing. For notational convenience, we shift π_D and write $\pi_D = (A_m, A_{m-1}, \ldots, A_0)$.

Let $S(A_i, t) = (t-1)C + (m-i)(2\epsilon + 3\delta)$, for $i = 0, \ldots, m$ and $t \in \mathbb{N}$.

We are now going to complete the proof of the Lemma by showing that $S(A_i, t)$ is a feasible schedule. We proceed by forward induction on t and backward induction on $i = m, m-1, \ldots, 0$. Assume that at the start of the first cycle all machines are loaded and have finished processing their part (this is without loss of generality, since we are only interested in the long run behavior of the system). For $t = 1$ and $i = m$, $S(A_m, 1) = 0$. For $t = 1$ and $i < m$, $S(A_i, 1) = (m-i)(2\epsilon + 3\delta)$, which is precisely the time required for the robot to perform $A_m, \ldots, A_{i+1}$, and to reach M_i.

Fix $t > 1$ and $i = m$; by induction, the robot arrives at M_m at time

$$\begin{aligned} S(A_0, t-1) + \epsilon + \delta + \epsilon + (m-1)\delta &= (t-2)C + m(2\epsilon + 3\delta) + 2\epsilon + m\delta \\ &\leq (t-2)C + C \\ &= (t-1)C \\ &= S(A_m, t). \end{aligned}$$

Thus the robot can reach M_m in time to perform A_m in the t-th cycle. In the previous cycle, the robot finished loading machine M_m at time

$$l(m, t-1) = S(A_{m-1}, t-1) + \epsilon + \delta + \epsilon.$$

We have:

$$\begin{aligned} S(A_m, t) - l(m, t-1) &= C - 2\epsilon - 3\delta - \epsilon - \delta - \epsilon \\ &= C - 4\epsilon - 4\delta \\ &\geq p_m. \end{aligned}$$

Thus machine M_m has finished processing the part at time $S(A_m, t)$ and can be unloaded.

Now, for $t > 1, i < m$: by induction, the robot starts unloading machine M_{i+1} at time $S(A_{i+1}, t)$. It then arrives at machine M_i at time $S(A_{i+1}, t) + \epsilon + \delta + \epsilon + 2\delta = S(A_i, t)$. In the

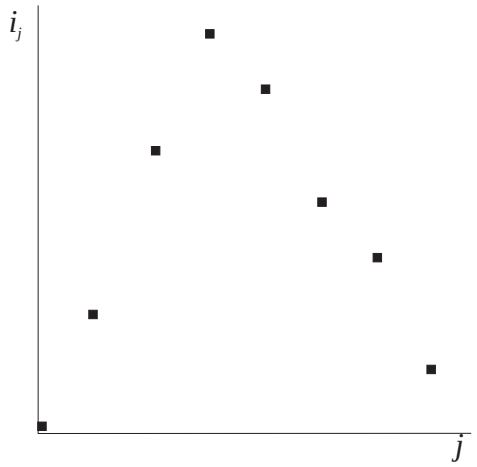

Figure 3: The pyramidal permutation $A_0, A_2, A_5, A_7, A_6, A_4, A_3, A_1$

previous cycle, it finished loading machine M_i at time $l(i, t-1) = S(A_{i-1}, t-1) + \epsilon + \delta + \epsilon$. This yields that

$$\begin{aligned} S(A_i, t) - l(i, t-1) &= C - 2\epsilon - 3\delta - \epsilon - \delta - \epsilon \\ &= C - 4\epsilon - 4\delta \\ &\geq p_i. \end{aligned}$$

Thus machine M_i has indeed finished processing at time $S(A_i, t)$, and the robot may start unloading. ■

Theorem 3.1. The optimal permutation π is such that:

$$\max\{2(m+1)(\delta+\epsilon), \max_i p_i + 4(\delta+\epsilon)\} \leq L(\pi) \leq \max\{4m\delta + 2(m+1)\epsilon, \max_i p_i + 4(\delta+\epsilon)\}.$$

Proof. The bounds follow from Lemma 3.1 and Lemma 3.2. ■

Incidentally, Theorem 3.1 implies that the cycle time of π_D is always smaller than twice the optimal cycle time. In other words, the algorithm that outputs π_D, independently of the values of the input parameters, is a 2-approximation algorithm for the identical parts cyclic scheduling problem! (We will not make use of this observation, but we find it interesting in its own right.) Moreover, π_D is optimal when $L(\pi_D) = \max_i p_i + 4(\delta + \epsilon)$. This provides an important proviso for the (unmotivated) claim made by Asfahl [1985, p. 274] that the permutation π_D 'must be held regardless of the relationship between the machine cycle times, the time required for the robot to move from station to station, and the load/unload times'. (the author calls 'machine cycle time' what we call 'processing time'.)

Definition 3.1. A set of permutations Π is *dominating* if for every choice of the processing times, there exists $\pi \in \Pi$ such that $L(\pi) \leq L(\pi')$ for all $\pi' \notin \Pi$.

We are now going to introduce a class of permutations, of which π_U and π_D are just two special representatives, and we are going to show that this class is dominating.

Let $\pi = (A_0, A_{i_1}, \ldots, A_{i_k}, A_{i_{k+1}}, \ldots, A_{i_m})$.

Definition 3.2. π is *pyramidal* if $1 \leq i_1 < \ldots < i_k = m$ and $m > i_{k+1} > \ldots > i_m \geq 1$.

In particular, the permutations π_U and π_D are pyramidal. The meaning of the adjective pyramidal should become clear from Figure 3. It is probably worth noticing that the concept of pyramidal permutations is not new : it has been introduced earlier, and extensively studied, in the literature on the traveling salesman problem; see Gilmore et al. [1985] for a thorough account, as well as Section 5 below. For an arbitrary, not necessarily pyramidal, permutation we also define:

Definition 3.3. Activity A_{i_k} is *uphill pyramidal* if there is an index l in $\{k, \ldots, m\}$ such that $i_k < i_j$ for all $k < j \leq l$, and $i_k > i_j$ for all $j < k$ and all $j > l$.

In words : all activities between A_{i_k} and A_{i_l} bear on machines located after M_{i_k} in the flowshop, while all activities before A_{i_k} or after A_{i_l} bear on machines located before M_{i_k}.

Definition 3.4. Activity A_{i_k} is *downhill pyramidal* if there is an index l in $\{0, \ldots, k\}$ such that $i_j > i_k$ for all $l \leq j < k$ and $i_j < i_k$ for all $j < l$ and all $j > k$.

In words : all activities between A_{i_l} and A_{i_k} occur on machines located after M_{i_k} in the flowshop, while all activities before A_{i_l} or after A_{i_k} occur on machines located before M_{i_k}.

Remarks.

1. A_0 and A_m are uphill pyramidal and A_m is downhill pyramidal in all permutations.
2. A permutation is pyramidal if and only if each activity is pyramidal (i.e. either uphill or downhill pyramidal) in this permutation.
3. The reader should convince himself that A_{i_k} is uphill pyramidal if and only if the trajectory $[M_{i_k}, M_{i_k+1}]$ is travelled exactly twice by the robot in each cycle : once when performing A_{i_k} and once after performing A_{i_l}.
4. Similarly, except for $i_k = m$, A_{i_k} is downhill pyramidal if and only if the trajectory $[M_{i_k}, M_{i_k+1}]$ is travelled exactly four times in each cycle : once just before A_{i_l}, once just before A_{i_k}, once during A_{i_k}, and once just after A_{i_k}.

The following theorem justifies our interest in pyramidal permutations. It will be the cornerstone for all subsequent results, and can therefore be viewed as the main result in this paper.

Theorem 3.2. The set of pyramidal permutations is dominating.

Proof. For reasons of clarity, and to stress that the theorem holds under very general conditions, we present the proof for the case where the machines are not necessarily equidistant, and loading/unloading times are machine dependent. We first introduce the following notations : for all $i, j = 0, \ldots, m$,

$$\delta_{ij} = \begin{cases} \sum_{k=i}^{j} \delta_k & \text{if } i \leq j \\ \\ \sum_{k=j}^{i} \delta_k & \text{if } j \leq i. \end{cases}$$

The time the robot takes to perform A_i is denoted by Δ_i:

$$\Delta_i = \epsilon_i + \delta_i + \epsilon_{i+1}.$$

Similarly to δ_{ij}, we define Δ_{ij} as:

$$\Delta_{ij} = \begin{cases} \sum_{k=i}^{j} \Delta_k & \text{if } i \leq j \\ \\ \sum_{k=j}^{i} \Delta_k & \text{if } j \leq i. \end{cases}$$

Let π be a nonpyramidal permutation. Let A_q be a nonpyramidal activity, let $A_{i_r} = A_b$ be the uphill pyramidal activity defined by $b = \max\{j | j < q \text{ and } A_j \text{ is uphill pyramidal}\}$, and let $A_{i_s} = A_e$ be the uphill pyramidal activity defined by $e = \min\{j | j > q \text{ and } A_j \text{ is uphill pyramidal}\}$.

Since A_{i_s} and A_{i_r} are uphill pyramidal, there exist indices i_l (associated with A_{i_s} as in Definition 3.3) and i_{k-1} (associated with A_{i_r} as in Definition 3.3) such that π can be rewritten in the form :

$$\pi = (A_0, \ldots, A_{i_r}, A_{i_{r+1}}, \ldots, A_{i_{s-1}}, A_{i_s}, \ldots, A_{i_l}, A_{i_{l+1}}, \ldots, A_{i_{k-1}}, A_{i_k}, \ldots, A_{i_m})$$

and

- all activities in $\pi_1 = (A_0, \ldots, A_{i_r})$ bear on machines with index at most $b+1$, i.e. $i_j \leq i_r = b$ for all A_{i_j} in π_1 (since A_{i_r} is uphill pyramidal),
- for all A_{i_j} in $\pi_2 = (A_{i_{r+1}}, \ldots, A_{i_{s-1}}), i_r = b < i_j < i_s = e$ (since A_{i_s} is uphill pyramidal),
- for all A_{i_j} in $\pi_3 = (A_{i_s}, \ldots, A_{i_l}), i_j \geq i_s = e$ (by definition of i_l),
- for all A_{i_j} in $\pi_4 = (A_{i_{l+1}}, \ldots, A_{i_{k-1}}), i_r = b < i_j < i_s = e$ (by definition of i_l and i_k),
- for all A_{i_j} in $\pi_5 = (A_{i_k}, \ldots, A_{i_m}), i_j < b$ (by definition of i_k).

Notice that π_1 and π_3 can never be empty since A_0 and A_m are uphill pyramidal by definition. Since there exists a nonpyramidal activity A_q, $\pi_2 \cup \pi_4$ cannot be empty, although one of π_2 or π_4 can. Finally notice that π_5 can be empty.

We claim that π is dominated by the new permutation

$$\pi' = \pi_1, \pi_3, A_{e-1}, A_{e-2}, \ldots, A_{b+1}, \pi_5,$$

i.e.

$$L(\pi') \leq L(\pi).$$

Before proving this claim, notice that the status (pyramidal or nonpyramidal) of all activities contained in π_1, π_3, π_5 is the same in π' as in π, and that all activities contained in $\pi_2 \cup \pi_4$, i.e. $A_{e-1}, \ldots, A_{b+1}$ are downhill pyramidal in π' (Figure 4 gives a sketchy representation of the permutation π'; where thick lines indicate the segments π_1, pi_3, π_5 that π' inherits from π). Thus the claim implies that, in at most m iterations, π can be transformed into a pyramidal permutation whose cycle time is no larger than that of π, which establishes Theorem 3.2.

Let a steady state schedule with minimum cycle time for π be given by $S(A_i, t)$. Denote by $l(i,t)$ the time at which the robot ends loading M_i in the *t-th* execution of the 1-unit cycle, for all $t \geq 1$, when it performs schedule S. We give now a steady state schedule $T(A_i, t)$ for π' such that $T(A_0, t) = S(A_0, t)$, thereby showing that the cycle time of π' is at most $L(\pi)$. We denote by $\lambda(i,t)$ the time at which the robot ends loading M_i in the *t-th* execution of the 1-unit cycle, for all $t \geq 1$, when it performs schedule T.

For all $t \geq 1$, we let

$$T(A_j, t) = S(A_j, t) \qquad \text{if } 0 \leq j \leq b \tag{1}$$

Next, for all $t \geq 1$ we define $T(A_{b+1}, t)$ by

$$T(A_{b+1}, t) = T(A_{i_k}, t) - \Delta_{b+1} - \delta_{(b+1)i_k} \qquad \text{if } \pi_5 \neq \emptyset \tag{2}$$

$$= T(A_0, t+1) - \Delta_{b+1} - \delta_{(b+1)0} \qquad \text{otherwise,} \tag{3}$$

and, recursively on j :

$$T(A_j, t) = T(A_{j-1}, t) - \Delta_j - \delta_{j(j-1)} \qquad \text{if } b+1 < j < e. \tag{4}$$

Finally , we let

$$T(A_{i_l}, t) = T(A_{e-1}, t) - \Delta_{i_l} - \delta_{i_l(e-1)} \tag{5}$$

and

$$T(A_j, t) = S(A_j, t) + T(A_{i_l}, t) - S(A_{i_l}, t) \qquad \text{if } e \leq j \leq m. \tag{6}$$

Notice that the definition is complete, i.e. $T(A_j, t)$ is defined for all $t \geq 1$ and for all $j \in \{0, \ldots, m\}$. In particular, (1) applies to π_1 and π_5 , (2)-(4) apply to π_2 and π_4 and (5)-(6) apply to π_3. One also checks easily that schedule T is steady state, with cycle time $L = L(\pi)$.

To prove that (1)-(6) define a feasible schedule for π', we need to check that:

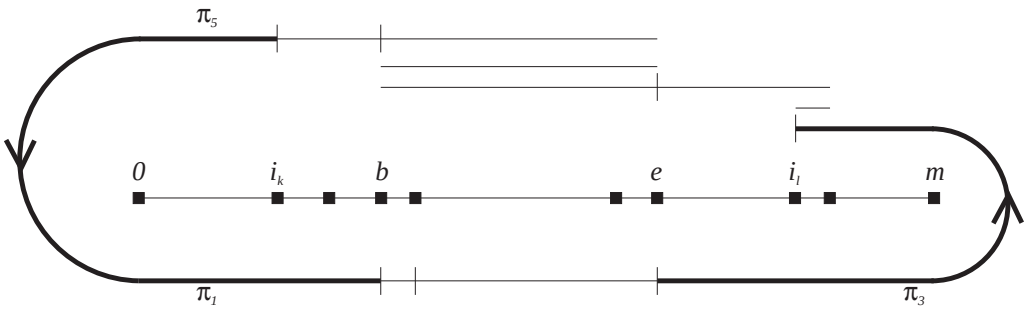

Figure 4: Graphical representation of the permutation π'

1. the robot can reach M_j before $T(A_j, t)$ in cycle t,
2. machine M_j has finished processing a part at time $T(A_j, t)$ in cycle t.

We first prove that the robot can reach all machines in time in every cycle. Consider any activity A_j, and let A_l be the activity preceding A_j in π'. If the start-time of A_l is defined by one of (2)-(5) (i.e. if $j \in \{b+1, \ldots, e-1\} \cup \{i_k\}$), then $T(A_j, t) - T(A_l, t)$ is exactly the time required for the robot to perform A_l (viz. Δ_l) and to subsequently move from M_{l+1} to M_j (viz. δ_{lj}). Thus the robot can get to M_j at time $T(A_j, t)$ if it can get to M_l at time $T(A_l, t)$.

The latter conclusion also applies if $0 \leq j \leq b, j \neq i_k$, in view of (2) and if $e < j \leq m$, in view of (6) (since the schedule S is feasible).

This reasoning leaves only open the question whether the robot can reach M_e at time $T(A_e, t)$ given that it starts with A_b (the activity preceding A_e in π') at time $T(A_b, t)$. Thus we have to check that

$$T(A_b, t) + \Delta_b + \delta_{(b+1)(e-1)} \leq T(A_e, t).$$

From the fact that in a schedule for π every trajectory $[M_j, M_{j+1}], b < j < e$ is travelled at least four times we can derive that (see fig 4.):

$$\begin{aligned} S(A_b, t+1) - S(A_b, t) \quad \geq \quad & S(A_b, t+1) - S(A_{i_k}, t) + S(A_{i_l}, t) - S(A_e, t) \\ & + \delta_{bi_k} + \delta_{i_l e} + \Delta_{i_l} + 3\delta_{(b+1)(e-1)} + \Delta_{(b+1)(e-1)} + \Delta_b. \end{aligned}$$

Combining this with (1) and (6) gives

$$\begin{aligned} T(A_b, t+1) - T(A_b, t) \quad \geq \quad & T(A_b, t+1) - T(A_{i_k}, t) + T(A_{i_l}, t) - T(A_e, t) \\ & + \delta_{bi_k} + \delta_{i_l e} + \Delta_{i_l} + 3\delta_{(b+1)(e-1)} + \Delta_{(b+1)(e-1)} + \Delta_b. \end{aligned}$$

Rewriting this inequality, we get

$$\begin{aligned} T(A_e,t) &\geq T(A_b,t) - T(A_{i_k},t) + T(A_{i_l},t) \\ &\quad +\delta_{(b+1)i_k} + \delta_{i_l e} + \Delta_{i_l} + 3\delta_{(b+1)(e-1)} + \Delta_{(b+1)(e-1)} + \Delta_b. \end{aligned}$$

Combining this with (2) and (4) leads to

$$\begin{aligned} T(A_e,t) &\geq T(A_b,t) - T(A_{e-1},t) + T(A_{i_l},t) \\ &\quad +\delta_{i_l(e-1)} + \Delta_{i_l} + \delta_{(b+1)(e-1)} + \Delta_b. \end{aligned}$$

and thus by (5)

$$T(A_e,t) \geq T(A_b,t) + \delta_{(b+1)(e-1)} + \Delta_b$$

as required.

Remark. Notice that we used A_{i_k}, which may not exist if π_5 is empty. In this case the result can be obtained similarly using $S(A_0, t+1)$ instead of $S(A_{i_k}, t)$.

Now we prove that machine M_j has indeed finished processing at time $T(A_j,t)$. By (1), all machines M_j with A_j in π_1 or π_5 are ready at time $T(A_j,t)$. Consider now machine M_{b+1}. Observe that the start of activity A_{b+1} in schedule T occurs as late as possible under the constraint that $S(A_{i_k},t) = T(A_{i_k},t)$ (see (1)-(3) and Figure 4). Thus, one derives that

$$S(A_{b+1},t) \leq T(A_{b+1},t)$$

and

$$T(A_{b+1},t) - T(A_b,t) \geq S(A_{b+1},t) - S(A_b,t).$$

Since $S(A_{b+1},t)$ is feasible, we have that

$$T(A_{b+1},t) - T(A_b,t) \geq S(A_{b+1},t) - S(A_b,t) \geq p_{b+1} + \Delta_b,$$

as required.

A straightforward extension of the argument used in Lemma 3.1 shows that

$$p_j + \Delta_j + \Delta_{j-1} + \delta_j + \delta_{j-1} \leq L, \text{ for all } j \in \{0,\ldots,m\}.$$

Thus, for all $b+1 < j < e$,

$$\begin{aligned} \lambda(j,t) &= T(A_{j-1},t) + \Delta_{j-1} \\ &= T(A_j,t) + \Delta_j + \delta_j + \delta_{j-1} + \Delta_{j-1} && \text{(by (4))} \\ &= T(A_j,t+1) + \Delta_j + \delta_j + \delta_{j-1} + \Delta_{j-1} - L, \end{aligned}$$

and thus

$$T(A_j, t+1) - \lambda(j,t) = L - (\Delta_j + \delta_j + \delta_{j-1} + \Delta_{j-1}) \geq p_j.$$

This is the required inequality : since A_j is a downhill activity, $T(A_j, t+1) - \lambda(j,t)$ represents the time elapsed between loading of a part in cycle t and its unloading in cycle $t+1$.

In view of (6), the machines A_j with $j > e$ create no problem. Finally, we have to check that M_e has finished processing in time:

$$\begin{aligned} T(A_e, t) - \lambda(e, t-1) &= T(A_e, t) - (T(A_{e-1}, t-1) + \Delta_{e-1}) \\ &= T(A_e, t) - (T(A_{i_l}, t-1) + \Delta_{i_l} + \delta_{i_l(e-1)} + \Delta_{e-1}) \qquad \text{(by (5))} \\ &= S(A_e, t) - (S(A_{i_l}, t-1) + \Delta_{i_l} + \delta_{i_l(e-1)} + \Delta_{e-1}). \qquad \text{(by (6))} \end{aligned}$$

Now, there are two cases.

1. If A_e precedes A_{e-1} in π (and thus the part loaded onto M_e in each execution of π is unloaded in the next execution):

   $$\begin{aligned} S(A_e, t) - (S(A_{i_l}, t-1) + \Delta_{i_l} + \delta_{i_l(e-1)} + \Delta_{e-1}) &\geq S(A_e, t) - (S(A_{e-1}, t-1) + \Delta_{e-1}) \\ &= S(A_e, t) - l(e, t-1), \end{aligned}$$

   and hence the feasibility of $T(A_e, t)$ follows from the feasibility of $S(A_e, t)$.

2. If A_{e-1} precedes A_e in π (and thus the part loaded onto M_e in each execution of π is unloaded in the same execution), it is not hard to see, by just checking the travel time that

   $$S(A_{e-1}, t) \geq S(A_{i_l}, t-1) + \Delta_{i_l} + \delta_{i_l(e-1)}.$$

   Hence

   $$\begin{aligned} T(A_e, t) - \lambda(e, t-1) &\geq S(A_e, t) - (S(A_{e-1}, t) + \Delta_{e-1}) \\ &= S(A_e, t) - l(e, t), \end{aligned}$$

   and again the feasibility of $T(A_e, t)$ follows from the feasibility of $S(A_e, t)$. ■

We remark that, when $m = 3$, there are exactly 4 pyramidal permutations, which have been proved by Sethi et al.[1992] to be dominating. Theorem 3.2 generalizes this result for arbitrary values of m.

4 An Algorithm for Computing the Cycle Time of a Pyramidal Permutation.

In this section we present an algorithm that computes a shortest steady state schedule for a pyramidal permutation in $O(m)$ time. This time complexity improves on the time complexity of the algorithm using the max-algebra approach [Cohen et al. 1985, Karp 1978], and on a related but faster algorithm based on the analysis in Van de Klundert [1994], and Karp [1978] (of course, the scope of our algorithm is also narrower).

While proving the correctness of the algorithm, we derive some structural properties of a shortest steady state schedule for a pyramidal permutation that will turn out to be useful in the next section.

Let $\pi = (A_0, A_{i_1}, \ldots, A_{i_m})$ be a pyramidal permutation of the activities, and let U resp. D denote the index set of the uphill, resp. downhill activities in π (with $m \in U \cap D$). A formal statement of our algorithm is given in Figure 5. We now discuss it more informally.

The algorithm computes a start time $S(A_i)$ for each activity A_i as well as a cycle time L_S. The schedule S is then implicitely defined by the relation

$$S(A_i, t) = S(A_i) + t \times L_S \text{ for } i = 0, \ldots, m \text{ and } t \in \mathbb{N}. \tag{7}$$

The algorithm proceeds backwards by decreasing activity index, starting with A_m. It schedules all downhill activities without waiting time, giving the robot just enough time to travel from machine to machine bewteen two activities. That is, if $i \in D$ and A_j is the downhill activity preceding A_i in π, then

$$S(A_i) = S(A_j) + (j + 1 - i)\delta + \delta + 2\epsilon. \tag{8}$$

Next, suppose that we are about to schedule an uphill activity A_i such that A_{i+1} is also uphill. Then, for every feasible schedule T, and for all $t \in \mathbb{N}$,

$$T(A_i, t) \leq T(A_{i+1}, t) - \delta - 2\epsilon - p_{i+1}, \tag{9}$$

and the algorithm simply sets

$$S(A_i) = S(A_{i+1}) - \delta - 2\epsilon - p_{i+1}. \tag{10}$$

Next consider an uphill activity A_i such that A_{i+1} is downhill. Again, in every feasible schedule T, and for all $t \in \mathbb{N}$,

$$T(A_i, t) \leq T(A_{i+1}, t) - \delta - 2\epsilon - p_{i+1}. \tag{11}$$

On the other hand, if A_j denotes the uphill activity following A_i in π, then we have in every feasible schedule T,

$$T(A_i, t) \leq T(A_j, t) - (j - i)\delta - 2\epsilon. \tag{12}$$

The algorithm takes (11) and (12) into account, and sets

$$S(A_i) = \min\{S(A_j) - (j-i)\delta - 2\epsilon, S(A_{i+1}) - \delta - 2\epsilon - p_{i+1}\}. \tag{13}$$

Observe that, if $S(A_i)$ is determined by the second term in the latter expression, then the difference $S(A_j) - S(A_i)$ is larger than the travel time required between A_i and A_j ; in other words the robot will have to incur some idle time before the execution of A_j.

In this way a starting time is determined for each activity. The cycle time L_S of the schedule, however, is still not determined. It can be seen that L_S must satisfy

$$S(A_0) + L_S \geq S(A_{i_m}) + (i_m + 2)\delta + 2\epsilon, \tag{14}$$

since otherwise, the robot cannot reach M_0 in time to start the (next) execution of A_0 after executing A_{i_m}. Moreover, by Lemma 3.1, we know that

$$L_S \geq \max_i p_i + 4(\delta + \epsilon). \tag{15}$$

Finally, consider any uphill activity A_i such that A_{i-1} is downhill. The part loaded on M_i in the t-th execution of A_{i-1} is unloaded from M_i in the $(t+1)$-st execution of A_i. Hence,

$$S(A_i) + L_S \geq S(A_{i-1}) + \delta + 2\epsilon + p_i. \tag{16}$$

In the algorithm, L_S is set to the minimum value that satisfies all three inequalities (14)-(16).

input : $\pi = (A_0, A_{i_1}, \ldots, A_{i_m})$

1. Set $S(A_m) = 0$. Set $i = m - 1$.
2. (Schedule A_i :)

   if $i \in D$ and A_j is the downhill activity preceding A_i in π then

   $$S(A_i) = S(A_j) + (j + 1 - i)\delta + \delta + 2\epsilon$$

   if $i \in U$ and $i + 1 \in U$ then

   $$S(A_i) = S(A_{i+1}) - \delta - 2\epsilon - p_{i+1}$$

   if $i \in U$ and $i + 1 \in D$ and A_j is the uphill activity following A_i in π then

   $$S(A_i) = \min\{S(A_j) - (j - i)\delta - 2\epsilon, S(A_{i+1}) - \delta - 2\epsilon - p_{i+1}\}$$

3. If $i > 0$ set $i \rightarrow i - 1$ and goto 2, else goto 4.
4. (Compute cycle time)

$$\begin{aligned} L_1 &= S(A_{i_m}) + (i_m + 2)\delta + 2\epsilon - S(A_0). \\ L_2 &= \max_i p_i + 4(\delta + \epsilon). \\ L_3 &= \max_{i \in U, i-1 \in D} S(A_{i-1}) + \delta + 2\epsilon + p_i - S(A_i) \\ L_S &= \max\{L_1, L_2, L_3\}. \end{aligned}$$

output : $\{S, L_S\}$

Figure 5: Algorithm for computing the cycle time of a pyramidal permutation

The algorithm can easily be implemented in $O(m)$ time. We now establish its correctness.

Theorem 4.1. For every pyramidal permutation π, the schedule defined by the algorithm in Figure 5 is feasible and has minimum cycle time among all schedules for π.

Proof. Feasibility of the schedule (7) can be checked by induction on i and t. In particular, for all $t \in \mathbb{N}$, $S(A_0, t+1)$ is feasible if $S(A_{i_m}, t)$ is feasible, because of (14). Moreover, if A_{i_j} starts at time $S(A_{i_j}, t)$, then the robot can reach machine $M_{i_{j+1}}$ before $S(A_{i_{j+1}}, t)$ (in time to perform $A_{i_{j+1}}$), because of (8), (10), (13). Finally, at time $S(A_i, t)$, machine M_i has finished processing and can be unloaded; this is true because of (10) if $i \in U$ and $i-1 \in U$; because of (16) if $i \in U$ and $i-1 \in D$; because of (11) if $i \in D$ and $i-1 \in U$; because of (8) and (15) if $i \in D$ and $i-1 \in D$. Thus the schedule defined by (7) is feasible.

It remains to show that the schedule defined by the algorithm in Figure 5 has minimum cycle time among all schedules for π. The following relation (17) is crucial for an intuitive understanding of the algorithm : it expresses that the time elapsed between the execution of an uphill activity A_u and a downhill activity A_d, is at least as short in S as in any other schedule.

We now claim the schedule S to have the following property : for every feasible schedule T, for all $t \in \mathbb{N}$, for all $u \in U$ and for all $d \in D$ such that either $u \geq d$ or $\{u, u+1, \ldots, d-1\} \subseteq U$,

$$T(A_d, t) - T(A_u, t) \geq S(A_d) - S(A_u). \tag{17}$$

We prove this by backward induction on u, for each fixed value of d. The claim holds for $u = m$, as follows easily from (8). Now suppose that it holds for $u = j$, and let A_i be the uphill activity immediately preceding A_j in π. If $j = i+1$, then (17) follows from (9), (10) and the induction hypothesis. If $j > i+1$, then A_{i+1} is downhill and $S(A_i)$ is given by (13). Now if,

$$S(A_i) = S(A_j) - (j-i)\delta - 2\epsilon.$$

then (17) follows from (12) and the induction hypothesis. On the other hand if

$$S(A_i) = S(A_{i+1}) - \delta - 2\epsilon - p_{i+1},$$

then, in view of (11),

$$T(A_{i+1}, t) - T(A_i, t) \geq S(A_{i+1}) - S(A_i)$$

for all $t \in \mathbb{N}$. Furthermore, since $i+1 \in D$, equation (8) implies

$$T(A_d, t) - T(A_{i+1}, t) \geq S(A_d) - S(A_{i+1}),$$

and (17) follows from the latter two inequalities. This completes the proof of the claim.

Let now T be any feasible schedule for π. Letting $u = 0$ and $d = i_m$ in the claim, we have in particular

$$T(A_{i_m}, t) - T(A_0, t) \geq S(A_{i_m}) - S(A_0)$$

for all $t \in \mathbb{N}$. Therefore,

$$\begin{aligned} T(A_0, t+1) - T(A_0, t) &\geq T(A_{i_m}, t) + (i_m + 2)\delta + 2\epsilon - T(A_0, t) \\ &\geq S(A_{i_m}) + (i_m + 2)\delta + 2\epsilon - S(A_0) \\ &= L_1. \end{aligned} \tag{18}$$

Next, consider an index $i \in U$ such that $i - 1 \in D$ and

$$L_3 = S(A_{i-1}) + \delta + 2\epsilon + p_i - S(A_i).$$

Letting $u = i$ and $d = i - 1$ in the claim, we obtain for all $t \in \mathbb{N}$:

$$T(A_{i-1}, t) - T(A_i, t) \geq S(A_{i-1}) - S(A_i).$$

Since T is feasible, the same reasoning that lead to (16) also establishes

$$T(A_i, t+1) \geq T(A_{i-1}, t) + \delta + 2\epsilon + p_i.$$

The previous inequalities together imply :

$$T(A_i, t+1) - T(A_i, t) \geq L_3. \tag{19}$$

From (18), (19) and Lemma 3.1, we now conclude that the long run average cycle time of T is at least $L_S = \max\{L_1, L_2, L_3\}$. This completes the proof of Theorem 4.1. ∎

5 Polynomial Algorithms for the Identical Parts Cyclic Scheduling Problem

Theorem 3.2 and Theorem 4.1 together imply that, for fixed m, the identical parts cyclic scheduling problem can be solved in constant time by enumerating all pyramidal permutations and subsequently computing their cycle time. However, since there are 2^{m-1} pyramidal permutations, the resulting algorithm has exponential complexity when m is considered to be part of the input. In this section, we will present more efficient algorithms, whose complexity grows only polynomially with m.

In the framework of the traveling salesman problem, a pyramidal tour of minimum length can be found by dynamic programming in $O(n^2)$ time, where n denotes the number of cities (see

e.g. Gilmore et al.[1985]). In terms of the identical parts cyclic scheduling problem, a shortest Hamiltonion tour would correspond to a permutation with minimum cycle time. Similarly, a shortest Hamiltonian path would correspond to a schedule in which $S(d) - S(0)$ is minimum, where d is the downhill activity with minimum index, i.e. the last activity in the permutation.

The first difficulty here stems from the fact that, in the traveling salesman problem, the distance between two cities is given explicitly in the distance matrix whereas in the identical parts cyclic scheduling problem, the 'distance' $S(A_{i_j}) - S(A_{i_{j+1}})$ between two consecutive activities is not a priori known, since the waiting time of the robot depends on the permutation. For the type of schedules constructed by the algorithm in the previous section, however, we will be able to show that these distances can somehow be computed online.

In this section, we first give a dynamic programming algorithm for the identical parts cyclic scheduling problem which computes, for every possible value of d, a pyramidal schedule S such that $S(d) - S(0)$ is minimum over all pyramidal schedules in which A_d is the downhill activity with minimum index. This dynamic programming algorithm is similar to the one computing a shortest *path* for the traveling salesman problem, but it does not necessarily output an optimal schedule (i.e. a *tour*) for the identical parts cyclic scheduling problem. This is the second difficulty encountered in our problem, in comparison with the traveling salesman problem. However, we show that, based on the dynamic programming formulation, an optimal schedule can be obtained in polynomial time.

Define now the following sets of permutations.

Definition 5.1. For all $u \in \{0, \ldots, m\}$ and $d \in \{1, \ldots, m\}$ with $u \neq d$, $\Pi_{u,d}$ is the set of pyramidal permutations such that:

1. A_u is uphill
2. A_d is downhill
3. if $u < d$, then A_i is uphill for all $i \in \{u, u+1, \ldots, d-1\}$
4. if $d < u$, then A_i is downhill for all $i \in \{d, d+1, \ldots, u-1\}$.

For the sake of simplicity, when $S(A_i, t)$ is a steady state schedule, we use the shorthand $S(A_i)$ instead of $S(A_i, 0)$ $(i = 1, \ldots, m)$.

Now we define a function $L(u, d)$ by:

Definition 5.2. For all $u \in \{0, \ldots, m\}$ and $d \in \{1, \ldots, m\}$ with $u \neq d$,

$$L(u,d) = \min\{ S_\pi(A_d) - S_\pi(A_u) | \pi \in \Pi_{u,d} \text{ and } S_\pi \text{ is a steady state schedule for } \pi\}$$

Theorem 5.1. For all $u \in \{0, \ldots, m\}$ and $d \in \{1, \ldots, m\}$ with $u \neq d$, the value of $L(u, d)$ can be computed in $O(m^2)$ time by the following dynamic programming formulation:

$$L(m-1, m) = \delta + 2\epsilon + p_m,$$

$$L(m, m-1) = 2\epsilon + 3\delta$$

and, for all $\{u,d\} \neq \{m-1,m\}$,

$$L(u,d) = \begin{cases} L(u,d+1) + 3\delta + 2\epsilon & \text{if } u > d+1 \\ \min_{j>u}\{L(u,j) + (j-d+2)\delta + 2\epsilon\} & \text{if } u = d+1 \\ L(u+1,d) + \delta + 2\epsilon + p_{u+1} & \text{if } u < d-1 \\ \min_{j>d}\{\max\{L(j,d) + 2\epsilon + (j-u)\delta, \delta + 2\epsilon + p_d\}\} & \text{if } u = d-1 \end{cases}$$

Proof. The expressions for $L(m-1,m)$ and $L(m,m-1)$ are easily checked to be correct (see (8) and (10)). For all other values of (u,d), the recursive equations are based on the algorithm given in the previous section (Figure 5). Their validity can be checked by induction. For example, assume that the value of $L(u,j)$ is correctly computed by these equations for all $j > u$, and consider next $L(u,u-1)$ (i.e., $u = d+1$). We must find a pyramidal permutation π and a corresponding schedule S which minimizes $S(A_{u-1}) - S(A_u)$. For any given permutation π, let A_j be the downhill activity immediately preceding A_{u-1} in π. From equation (8), we know that

$$S(A_{u-1}) = S(A_j) + (j-u+3)\delta + 2\epsilon.$$

Moreover, relying on the dynamic programming principle of optimality, we can assume that $S(A_j) - S(A_u)$ is as small as possible under the previous restrictions, i.e. $S(A_j) - S(A_u) = L(u,j)$. It follows now that

$$\begin{aligned} S(A_{u-1}) - S(A_u) &= L(u,j) + (j-u+3)\delta + 2\epsilon \\ &= L(u,j) + (j-d+2)\delta + 2\epsilon. \end{aligned}$$

Thus, $L(u,u-1)$ is attained by a permutation π which minimizes the previous expression, as is asserted in the statement of the theorem. The other cases are left to the reader.

As for the complexity of the formulation, notice that the value of each $L(u,d)$ with $|u-d| \geq 2$ can be computed in constant time. The computation of each $L(u,d)$ wiht $|u-d| = 1$ requires $O(m)$ time, but there are only $2m$ pairs (u,d) such that $|u-d| = 1$. Thus all values $L(u,d)$ can be obtained in $O(m^2)$ time. ■

The dynamic programming formulation in Theorem 5.1 allows to compute in $O(m^2)$ time, for every possible last activity $A_d, d = 1, \ldots, m$:

- the value of $L(0,d)$
- a permutation $\pi_d \in \Pi_{0,d}$.
- a schedule S_{π_d} for π_d such that $S_{\pi_d}(A_d) - S_{\pi_d}(A_0) = L(0,d)$.

The schedule S_{π_d} is the same schedule that would have been output by the algorithm in Figure 5, had it taken π_d as input. It follows then that the cycle time of the permutation π_d produced by the dynamic programming algorithm can be computed as in step 4 of the algorithm in Figure 5. But again we emphasize here that the permutation π_d output by the dynamic programming algorithm does not necessarily have minimum cycle time. In the remainder of this section, we explain how the dynamic programming formulation can be used to solve the identical parts cyclic scheduling problem to optimality.

Let us first focus on the recognition version of the problem, which may be stated as :

Input : $p_i, i = 1, \ldots, m, \delta, \epsilon, C$

Question : *Is there a steady state schedule with cycle time at most C ?*

This problem can be solved by a slight adaption of the dynamic programming algorithm. Informally, the dynamic programming algorithm will be modified so that, when it finds a permutation, then the cycle time of the permutation is less than or equal to C, and when it does not find a permutation, then such a permutation does not exist.

To start with, let us assume from now on that $\max_i p_i + 4(\delta + \epsilon) \leq C$, since otherwise Theorem 3.1 provides a negative answer to the recognition problem. Consider next the following definition, motivated by the computation of the bound L_3 in Figure 5 :

Definition 5.3. For all $u \in \{0, \ldots, m\}$ and $d \in \{1, \ldots, m\}$ with $u \neq d$,

$$
\begin{aligned}
L_C(u,d) \quad &= \quad \min S_\pi(A_d) - S_\pi(A_u) \\
&\\
&\text{s.t.} \qquad \pi \in \Pi_{u,d} \\
&\qquad\quad\; S_\pi \text{ is a steady state schedule for } \pi \\
&\qquad\quad\; \max_{i \in U, i \geq u, i-1 \in D, i-1 \geq d} \{S_\pi(A_{i-1}) + \delta + 2\epsilon + p_i - S_\pi(A_i)\} \leq C.
\end{aligned}
$$

We let $L_C(u,d) = +\infty$ if there is no feasible solution to the optimization problem in Definition 5.3. Notice that $L_C(0,d) < +\infty$ for all $d \in \{1, \ldots, m\}$, since the permutation $(A_0, A_1, \ldots, A_{d-1}, A_m, A_{m-1}, \ldots, A_d)$ admits a schedule which satisfies all constraints in the definition of $L_C(0,d)$. It can be checked as in Theorem 5.1 that the values $L_C(u,d)$ can be computed in $O(m^2)$ time by the following recursion (where, for the sake of compactness, we denote by $K(u,d)$ the quantity $\min_{j>u}\{L_C(u,j) + (j-d+2)\delta + 2\epsilon\}$) :

$$
\begin{aligned}
L_C(m-1,m) \quad &= \quad \delta + 2\epsilon + p_m, \\
&\\
L_C(m,m-1) \quad &= \quad 2\epsilon + 3\delta
\end{aligned}
$$

and, for all $\{u,d\} \neq \{m-1,m\}$,

$$L_C(u,d) = \begin{cases} L_C(u,d+1) + 3\delta + 2\epsilon & \text{if } u > d+1 \\ K(u,d) & \text{if } u = d+1 \text{ and } K(u,d) + \delta + 2\epsilon + p_u \leq C \\ +\infty & \text{if } u = d+1 \text{ and } K(u,d) + \delta + 2\epsilon + p_u > C \\ L_C(u+1,d) + \delta + 2\epsilon + p_{u+1} & \text{if } u < d-1 \\ \min_{j>d}\{\max\{L_C(j,d) + 2\epsilon + (j-u)\delta, \delta + 2\epsilon + p_d\}\} & \text{if } u = d-1 \end{cases}$$

Theorem 5.2. The recognition version of the identical parts cyclic scheduling problem can be solved in $O(m^2)$ time.

Proof. We can compute in $O(m^2)$, for each $d \in \{1,\ldots,m\}$:

- the value of $L_C(0,d)$
- a permutation $\pi_d \in \Pi_{0,d}$.
- a schedule S_{π_d} for π_d such that $S_{\pi_d}(A_d) - S_{\pi_d}(A_0) = L(0,d)$ and S_{π_d} satisfies the third constraint in Definition 5.3.

We claim that the answer to the recognition problem is affirmative if and only if there exists $d \in \{1,\ldots,m\}$ such that

$$L_C(0,d) + (d+2)\delta + 2\epsilon \leq C. \tag{20}$$

Indeed, if (20) holds for some d, then the cycle time of π_d is at most C (see Step 4 in Figure 5), and we are done. Conversely, assume that there exists a pyramidal permutation, say π, whose cycle time is at most C. Let A_d be the last downhill activity in π, and let S_π be the schedule computed for π by the algorithm in Figure 5. By Definition 5.3, $L_C(0,d) \leq S_\pi(A_d) - S_\pi(A_0)$. Moreover, in view of step 4 in Figure 5, $S_\pi(A_d) - S_\pi(A_0) + (d+2)\delta + 2\epsilon \leq C$. Thus we conclude that (20) holds, which concludes the proof. ■

Of course, the optimization version of the problem can be solved by repeatedly solving the recognition version, while applying binary search between the lowerbound and the upperbound given in Theorem 3.2 :

Corollary 5.1. For integral values of $p_i, i = 1,\ldots,m,\ \delta, \epsilon$ the optimization version of the identical parts m-machine cyclic scheduling problem can be solved in $O(m^2 \log(m\delta))$ time.

In the last part of this section, we now describe a strongly polynomial algorithm to solve the optimization version of the identical parts m-machine cyclic scheduling problem. We first need yet another modification of Definition 5.2, in which some activities are 'forced' to be downhill

(the motivation for this definition should become clear very shortly).

Definition 5.4. For all $F \subseteq \{1, \ldots, m\}$, $u \in \{0, \ldots, m\} \setminus F$ and $d \in \{1, \ldots, m\}$ with $u \neq d$,

$$\begin{aligned} L_F(u,d) \quad &= \quad \min S_\pi(A_d) - S_\pi(A_u) \\ \\ &\text{s.t.} \qquad \pi \in \Pi_{u,d} \\ &\qquad\quad\; S_\pi \text{ is a steady state schedule for } \pi \\ &\qquad\quad\; A_i \text{ is downhill in } \pi, \text{ for all } i \in F. \end{aligned}$$

For any $F \subseteq \{1, \ldots, m\}$, a straightforward adaption of our previous dynamic programming algorithm, which simply 'skips' all pairs (u, d) such that $u \in F$, allows to compute in $O(m^2)$ time, for each $d \in \{1, \ldots, m\}$:

- the value of $L_F(0, d)$
- a permutation $\pi_{F,d}$ such that A_i is downhill in $\pi_{F,d}$ for all $i \in F$
- a schedule $S_{F,d}$ for $\pi_{F,d}$ such that $S_{F,d}(A_d) - S_{F,d}(A_0) = L_F(0, d)$.

The cycle time of $\pi_{F,d}$ (as computed by the algorithm in Figure 5) is denoted by $L(\pi_{F,d})$. Suppose now that $L(\pi_{F,d}) = L_3$. We call activity A_i an *obstruction* of $\pi_{F,d}$ if A_i is uphill in $\pi_{F,d}$, A_{i-1} is downhill in $\pi_{F,d}$ and $L(\pi_{F,d}) = S_{F,d}(A_{i-1}) + \delta + 2\epsilon + p_i - S_{F,d}(A_i)$. Roughly speaking, the intuition behind the algorithm that we are about to present is that, if a current schedule is not optimal, then it must contain an obstruction A_i, and A_i should be downhill in any optimal schedule. This property is stated more precisely in the following two lemmas.

Lemma 5.1. For all $F \subseteq \{1, \ldots, m\}$ and $d \in \{1, \ldots, m\}$, if there is no obstruction in $\pi_{F,d}$, then there is no pyramidal permutation with cycle time less than $L(\pi_{F,d})$ in which all activities in F are downhill and A_d is the last downhill activity.

Proof. If $L(\pi_{F,d}) = \max_{1 \leq i \leq m} p_i + 4(\delta + \epsilon)$, then $\pi_{F,d}$ is optimal by Theorem 3.1. If this is not the case, and there is no obstruction in $\pi_{F,d}$, then by definition of the bound L_1 in Figure 5

$$T(A_0, t+1) - T(A_d, t) \geq S_{F,d}(A_0, t+1) - S_{F,d}(A_d, t)$$

for every feasible schedule T. Combined with the definition of $L_F(0, d)$, this proves the lemma. ■

Lemma 5.2. For all $F \subseteq \{1, \ldots, m\}$ and $d \in \{1, \ldots, m\}$, if A_i is any obstruction in $\pi_{F,d}$, then there is no pyramidal permutation with cycle time less than $L(\pi_{F,d})$ in which all activities in F are downhill and A_i is uphill.

Proof. Let π be any pyramidal permutation in which all activities in F are downhill and A_i is uphill, and let T be a shortest steady state schedule for π. Suppose first that A_{i-1} is downhill in π. Notice that, by definition,

$$L_F(i, i-1) = S_{F,d}(A_{i-1}) - S_{F,d}(A_i) \leq T(A_{i-1}, t) - T(A_i, t).$$

On the other hand, $\delta + 2\epsilon + p_i$ is a lowerbound on $T(A_i, t+1) - T(A_i, t)$. Thus

$$\begin{aligned} L(\pi_{F,d}) &= S_{F,d}(A_{i-1}) + \delta + 2\epsilon + p_i - S_{F,d}(A_i) \\ &\leq T(A_i, t+1) - T(A_i, t) \end{aligned}$$

as required.

Next, suppose that both A_i and A_{i-1} are uphill in π. Then,

$$\begin{aligned} T(A_i, t+1) - T(A_{i-1}, t+1) &\geq \delta + 2\epsilon + p_i \\ &= S_{F,d}(A_i, t+1) - S_{F,d}(A_{i-1}, t). \end{aligned}$$

Now consider the first point after $T(A_i, t)$, say τ, at which the robot reaches M_{i-1} after travelling trajectory (M_i, M_{i-1}). By definition of $L_F(i, i-1)$,

$$\tau - T(A_i, t) \geq S_{F,d}(A_{i-1}, t) - S_{F,d}(A_i, t)$$

because the permutation π', obtained by switching the status of A_{i-1} from uphill to downhill in π, admits a shortest schedule T' such that $T'(A_i, t) = T(A_i, t)$ and $T'(A_{i-1}, t) = \tau$, as implied by the algorithm in Figure 5. Combining the latter inequalities one derives that

$$\begin{aligned} L(\pi) &= T(A_i, t+1) - T(A_i, t) \\ &\geq T(A_i, t+1) - T(A_{i-1}, t+1) + \tau - T(A_i, t) \\ &\geq S_{F,d}(A_i, t+1) - S_{F,d}(A_{i-1}, t) + S_{F,d}(A_{i-1}, t) - S_{F,d}(A_i, t) \\ &= L(\pi_{F,d}) \end{aligned}$$

as required. ■

Combining Lemma 5.1. and Lemma 5.2. we obtain the following result :

Theorem 5.3. The optimization version of the identical parts m-machine cyclic scheduling problem can be solved in $O(m^3)$ time.

Proof. We claim that the following algorithm correctly solves the problem:

1. $F \rightarrow \emptyset$, $opt \rightarrow +\infty$
2. Call the dynamic programming algorithm and compute $L(\pi_{F,d})$, $\pi_{F,d}$, $S_{F,d}$ for $d = 1, \ldots, m$.
3. Select d such that

   $$S_{F,d}(A_d) - S_{F,d}(A_0) + (d+2)\delta + 2\epsilon = \min_j \{S_{F,j}(A_j) - S_{F,j}(A_0) + (j+2)\delta + 2\epsilon\}$$

   $opt \rightarrow \min(opt, L(\pi_{F,d}))$.

4. If $opt = \max_i p_i + 4(\delta + \epsilon)$, return $\pi_{F,d}$ and stop.

5. (Lemma 5.1.) If there are no obstructions in $\pi_{F,d}$, return a permutation with cycle time equal to *opt*, and stop.

6. (Lemma 5.2.) Let I be the set of obstructions in $\pi_{F,d}$. Set $F \rightarrow F \cup I$ and goto 2.

Observe that the complexity of this algorithm is indeed $O(m^3)$ since $|F| \leq m$, and hence the dynamic programming algorithm cannot be called more than m times.

To see that the algorithm is correct, assume first that the following property (P) holds before some iteration of Step 2 : (P) if there is a permutation, say π, with cycle time smaller than *opt*, then all activities in F are downhill in π (notice that property (P) certainly holds before the first iteration of Step 2.) Under this assumption, we are going to prove that property (P) is an invariant of the algorithm, i.e. : if property (P) holds before some iteration of Step 2, then either the algorithm returns an optimal value in the subsequent execution of Steps 4-5, or property (P) holds again before the next iteration of Step 2.

Indeed, if the algorithm stops in Step 4, then $\pi_{F,d}$ is optimal by Lemma 3.1. Suppose now that it stops in Step 5, and (by contradiction) that there exists a permutation π with cycle time $L(\pi) < opt$. Let S_π be any schedule for π, and let A_j be the last downhill activity in π. By the property (P), all activities in F are downhill in π. Hence by Definition 5.4 and by definition of $S_{F,j}$:

$$S_\pi(A_j) - S_\pi(A_0) \geq S_{F,j}(A_j) - S_{F,j}(A_0).$$

Moreover,

$$L(\pi) \geq S_\pi(A_j) - S_\pi(A_0) + (j+2)\delta + 2\epsilon,$$

and thus

$$L(\pi) \geq S_{F,j}(A_j) - S_{F,j}(A_0) + (j+2)\delta + 2\epsilon.$$

Step 3 of the algorithm implies now:

$$L(\pi) \geq S_{F,d}(A_j) - S_{F,d}(A_0) + (d+2)\delta + 2\epsilon.$$

Since $\pi_{F,d}$ has no obstruction, the previous inequality boils down to

$$L(\pi) \geq L(\pi_{F,d}),$$

which contradicts $L(\pi) < opt$.

Finally if the algorithm does not stop in either Step 4 or 5, then $\pi_{F,d}$ must contain a set of obstructions, denoted by I. Setting $F \rightarrow F \cup I$, property (P) is now directly implied by Lemma 5.2.

Thus property (P) is indeed an invariant of the algorithm, and we conclude that the algorithm is correct (since it is finite). ■

6 Summary and Directions for Future Research

Planning and scheduling in modern production environments, such as robotic cells, gives rise to a variety of challenging decision problems that do not fit well into classical models. In this paper, we have studied a throughput rate maximization problem in a flowshop-like robotic cell in which the material handling system consists of a single robot or robot arm. The throughput rate of the cell is highly dependent on the interaction between the material handling system and the machines processing the parts. We have shown that, when there is only one type of parts to be produced, the problem can be solved in (strongly) polynomial time even if the number of machines is viewed as an input parameter of the problem. This generalizes previous results established by Sethi et al. [1992] for the three-machine case. Interestingly, our analysis makes heavy use of seemingly unrelated concepts and techniques investigated by various authors in connection with the traveling salesman problem (although, it should be observed, we never actually obtain a TSP-formulation of our problem).

Many interesting related problems are still open. The first open problem we mention is the conjecture of Sethi et al. [1992] that 1-unit cycles are optimal among all possible robot move sequences, in the case where there is only one part-type to be produced. Other interesting open problems concern the case where there is more than one part-type. The applicability of the concept of pyramidal permutations to such situations seems to be limited for a number of reasons. First of all, there exist problem instances with multiple part types in which 1-unit cycles can be shown to be dominated (see Hall et al. [1993a]). Second, even if we restrict the analysis to 1-unit cycles, it is not clear whether there always exists an optimal permutation that is pyramidal. Finally, an NP-hardness result of Hall et al. [1993b] (mentioned in the introduction) establishes that computing the optimal part input sequence in a three machine robotic cell is NP-hard for the downhill permutation, and thus for pyramidal permutations in general. We also notice that the complexity of the multiple parts problem remains open if either the number of parts or the number of part-types is fixed. As a matter of fact, to the best of our knowledge, this question appears to be open even for ordinary three machine flowshops (without robot). Related issues have been recently investigated by Agnetis [1989], Hochbaum & Shamir[1991], Granot et al.[1993], etc.

7 Acknowledgements

The authors have benefited from helpful discussions with Jaap Geerdink and Olaf Flippo. We are grateful to the Associate Editor and the reviewers of this paper for their helpful comments. The first author was partially supported in the course of this research by AFOSR (grant F49620-93-1-0041) and ONR (grants N00014-92-J1375 and N00014-92-4083).

8 References

Agnetis, A. 1989. No-wait flow shop scheduling with large lot size. Rap. 16.89. Universita degli studi di Roma 'La Sapienza'.

Agnetis, A., Lucertini, M. & Nicolo, F. 1993. Flow management in flexible manufacturing cells

with pipeline operations. Management Science Vol. 39, No. 3, 294-306.

Asfahl, C.R. 1985. Robots and Manufacturing Automation, John Wiley & Sons, New York, NY.

Blazewicz, J., Eiselt, H.A., Finke, G., Laporte, G. & Weglarz, J. 1991. Scheduling tasks and vehicles in a flexible maufacturing system. International Journal of Flexible Manufacturing Systems 4, 5-16.

Cohen, G., Dubois, D., Quadrat, J.P. & Viot, M. 1985. A linear-system-theoretic review of discrete-event processes and its use for performance evaluation in manufacturing. IEEE Transactions on Automatic Control, Vol AC 30, No 3, 210-220.

Gilmore, P.C., Lawler, E.L. & Shmoys D.B. 1985. Well solved special cases in Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.) The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons. Chichester New York Brisbane Toronto Singapore.

Granot, F., Skorin-Kapov, J. & Tamir, A. 1993. Using quadratic programming to solve high multiplicity scheduling problems on parallel machines. Working Paper.

Hall, N.G., Kamoun, H. & Sriskandarajah, C. 1993a. Scheduling in robotic cells: Two machine cells and identical parts. Working Paper, College of Business, The Ohio State University. To appear in Operations Research.

Hall, N.G., Kamoun, H. & Sriskandarajah, C. 1993b. Scheduling in robotic cells: Large cells. Working Paper, College of Business, The Ohio State University.

Hall, N.G., Potts, C.N. & Sriskandarajah, C. 1994. Parallel machine scheduling with a common server. Working Paper, College of Business, The Ohio State University.

Hochbaum, D.S. & Shamir, R. 1991. Strongly polynomial algorithms for the high multiplicity scheduling problem. Operations Research Vol. 39, No. 4, 648-653.

Jeng, W.D., Lin, J.T. & Wen, U.P. 1993. Algorithms for sequencing robot activities in a robot-centered parallel-processor workcell. Computers & Operations Research Vol. 20, No. 2, 185-197.

Karabati, S. & Kouvelis, P. 1993. Cyclic scheduling in flow lines: modeling observations, effective heuristics and optimal cycle time minimization procedure. Working paper.

Karp, R.M. 1978. A characterization of the minimum cycle mean in a digraph. Discrete Mathematics 23, 309-311.

King, R.E., Hodgson, T.J. & Chafee, F.W. 1993. Robot task scheduling in a flexible manufacturing cell. IIE Transactions Vol. 25, No. 2, 80-87.

Kise, H. 1991. On an automated two-machine flowshop scheduling problem with infinite buffer.

Journal of the Operations Research Society of Japan Vol. 34, No. 3, 354-361.

Kise, H., Shioyama, T. & Ibaraki, T. 1991. Automated two machine flowshop scheduling: a solvable case. IIE Transactions Vol. 23, No 1, 10-16.

Krishnamurthy, N.N., Batta, R. & Karwan, M.H. 1993. Developping conflict-free routes for automated guided vehicles. Operations Research Vol. 41, No. 6, 1077-1090.

McCormick, S.T., Pinedo, M.L., Shenker, S. & Wolf, B. 1989. Sequencing in an assembly line with blocking to minimize cycle time. Operations Research Vol 37, No 6, 925-935.

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

Stecke, K.E. 1983, Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems. Management Science Vol. 29, No. 3, 273-288.

Van de Klundert, J.J. 1994, Ph.D. Thesis in preparation.

Appendix

We show here that, among all schedules with maximum long run throughput rate for the identical parts cyclic scheduling problem, there always exists one which is steady state.

Let π be permutation of the activities. We construct a directed graph $D(V, A)$ as follows. Associate a vertex v_i with each activity A_i, for all $i = 0, \ldots, m$. There is an arc from vertex v_i to vertex v_j if $i = \pi(s), j = \pi(s+1)$, for some $s \in 0, \ldots m$ or if $i = \pi(m)$ and $j = \pi(0)$. This arc represents the fact that the robot executes activity A_j right after activity A_i. The length of this arc is the time the robot takes to perform A_i and subsequently travel to machine M_j to perform A_j. Furthermore there is an arc with length $\delta + 2\epsilon + p_{i+1}$ from vertex v_i to vertex v_{i+1}, $i = 0, \ldots, m-1$.

Observe that there are two arcs from v_i to v_j if $j = i+1$ and $i = \pi(s), j = \pi(s+1)$, for some $s \in 0, \ldots m$. In such a case the shortest of these two arcs (i.e. the robot travel time arc) is deleted.

After this modification D is a directed cyclic graph. Loosely speaking, the length of a path from v_i to v_j in D is a lowerbound on the time that the execution of A_i and A_j must be apart. More precisely, denote by $L(v, u, t)$ the length of the longest path from v to u, $u, v \in V$ that goes around D t times, $t \in \mathbb{N}$. It is not hard to see that :

Lemma A.1. Let T be any real-valued function on $\{(A_i, t) | i = 0, \ldots, m, t \in \mathbb{N}\}$. Then T is a schedule for π if and only if,

$$L(v_i, v_j, t) \leq T(A_j, t) - T(A_i, 0)$$

if A_i precedes A_j in π, $i, j \in \{0, \ldots, m\}, t \in \mathbb{N}$, and

$$L(v_i, v_j, t-1) \leq T(A_j, t) - T(A_i, 0),$$

if A_j precedes A_i in π, $i, j \in \{0, \ldots, m\}, t \in \mathbb{N}$.

Proof. Trivial.

In particular, the length of a path from v_i to v_i that goes around D once is a lowerbound on the cycle time of π. More generally, denote by C the shortest possible cycle time of the 1-unit cycle under consideration.

$$C \geq \max_{t \in \mathbb{N}} \max_{0 \leq i \leq m} \frac{L(v_i, v_i, t)}{t} \equiv L^*. \tag{21}$$

Notice that any cycle that goes around D more than m times must contain twice the same vertex, by the pigeonhole principle, and hence can be split into several simple cycles. Therefore we have,

$$L^* = \max_{1 \leq t \leq m+1} \max_{0 \leq i \leq m} \frac{L(v_i, v_i, t)}{t} \tag{22}$$

For the moment, let us not concentrate on how to find the maximum in (22), but rather on how to construct a feasible steady state schedule when this maximum is known. Suppose that v^* and t^* realize the maximum in (22), i.e.

$$L^* = \frac{L(v^*, v^*, t^*)}{t^*}.$$

For $i = 0, \ldots, m$, we define

$$S(A_i, 0) = \max_{0 \leq t \leq m} \{L(v^*, v_i, t) - t \times L^*\}, \tag{23}$$

$$S(A_i, s) = S(A_i, 0) + s \times L^*, \text{ for all } s \in \mathbb{N}. \tag{24}$$

Lemma A.2. S is a steady state schedule for π with cycle time L^*.

Proof. Shift π such that the activity corresponding to v^* is the first activity in π. Let A_i and A_j be arbitrary activities in π. We deal here with the case that A_i precedes A_j in π (the other case being left to the reader). According to Lemma A.1, we must show in this case that S satisfies

$$L(v_i, v_j, t) \leq S(A_j, 0) + t \times L^* - S(A_i, 0), \tag{25}$$

$t \in \mathbb{N}$. In fact, we only have to establish (25) for $t \in \{0, \ldots, m\}$. Indeed, if $t > m$ there is at least one vertex w that appears twice on the longest path from v_i to v_j. Let t' be the number of times the path goes around D between these two occurrences of w. By definition of L^*, $L(w, w, t') \leq t' \times L^*$. Hence it suffices to show that

$$L(v_i, v_j, t - t') \leq (t - t') \times L^* + S(A_j, 0) - S(A_i, 0).$$

Using induction, we deduce that only values of t in $\{0, \ldots, m\}$ need be considered.

Now, choose $s, 0 \leq s \leq m$, such that

$$S(A_i, 0) = L(v^*, v_i, s) - s \times L^*, \tag{26}$$

and observe that

$$L(v^*, v_i, s) + L(v_i, v_j, t) \leq L(v^*, v_j, t + s). \tag{27}$$

Now, we claim that

$$L(v^*, v_j, t + s) \leq S(A_j, 0) + (t + s) \times L^*. \tag{28}$$

Indeed, in case $s + t \leq m$, this is true by definition of S. In case $s + t > m$ we have again that there exists a vertex w that appears twice on the longest path from v^* to v_j. Let t' be the number of times the path goes around between these two occurrences of w. By definition of L^*, $L(w, w, t') \leq t' \times L^*$. Hence, (28) follows by induction from

$$L(v^*, v_j, t - t') \leq S(A_j, 0) + (t + s - t') \times L^*.$$

Combining (27) and (28), we derive

$$L(v^*, v_i, s) + L(v_i, v_j, t) \leq L(v^*, v_j, t + s) \leq S(A_j, 0) + (t + s) \times L^*. \tag{29}$$

Hence, from (26) and (29),

$$L(v_i, v_j, t) \leq S(A_j, 0) + t \times L^* - S(A_i, 0),$$

as required. ■

Theorem A.1. For each permutation of the activities, there exists a steady state schedule that maximizes the long run throughput rate.

Proof. This follows from Lemma A.2 and (21). ■

As a final remark, we observe that, for each permutation π of the activities, there exists a natural 'active' schedule, in which the robot performs each activity as early as possible, in the order dictated by π. It is not very difficult to see that the long run average throughput rate of an active schedule is always optimal for the corresponding permutation (see also Cohen et al. [1985]).