IIE Transactions (2000) **32**, 449–459

Scheduling operations on parallel machine tools

BRYAN A. NORMAN[1,*] and JAMES C. BEAN[2]

[1]*Department of Industrial Engineering, University of Pittsburgh, Pittsburgh, PA 15261, USA*
E-mail: banorman@engrng.pitt.edu
[2]*Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI 48109, USA*
E-mail: jbean@umich.edu

Accepted June 1998

We introduce unique scheduling problems that arise for multiple spindle machine tools. The ability of these machines to perform simultaneous operations on more than one part creates constraints that are not found in the traditional scheduling literature. Two types of solution procedures are introduced for these problems. The first uses priority dispatching rules and a delay factor concept, while the second uses a genetic algorithm with a random keys encoding. The effectiveness of these methods is demonstrated on test problems with comparisons to lower bounds.

1. Introduction to parallel machine tools

Machining hardware advances drive changes in requirements for Computer-Aided Process Planning (CAPP) systems. To gain the full benefit of improvements in hardware, CAPP software that can exploit these improvements must be developed. A key difference between Parallel Machine Tools (PMTs) and conventional CNC machines is that the former contain multiple spindles and can hold multiple workpieces concurrently. As a result, a PMT can process more than one workpiece at a time and/or perform more than one operation at a time. This violates the most basic assumptions of traditional scheduling or process planning.

To properly describe PMTs it is necessary to define some terms. We retain the terminology introduced in Levin and Dutta [1]. A *Part Machining Location* (PML) refers to a valid workholding location. The main spindle and subspindle(s) always represent valid PMLs. A *Machining Unit* (MU) refers to a tool holding device, which may hold a single tool or a turret containing multiple tools. Relative motions between the tool on the MU and the workpiece held in the PML accomplish the machining. Conventional machines have only one MU and one PML. PMTs have $PML_{\max}(\geq 1)$ PMLs and $MU_{\max}(\geq 1)$ MUs. $PML_{\max}$ indicates the maximum number of workpieces on the machine at one time, and $MU_{\max}$ the maximum number of operations being carried out simultaneously. Note that this is much more general than traditional machines that, with lockstepped PMLs and MUs, can make multiple identical copies of the same part simultaneously. For a more detailed discussion of the structure of PMTs, see Levin and Dutta [1].

Due to the presence of multiple PMLs and multiple MUs, PMTs offer new challenges for process planning systems. Most of the existing process planning and scheduling literature assumes that machines can process only one part at a time and that only one operation can be performed on a part at a time. However, PMTs are not limited by these assumptions.

The scheduling of operations on a PMT has not received much attention in the literature. Two papers that discuss process planning for PMTs, Levin and Dutta [1] and Yip-Hoi and Dutta [2], mention the importance of scheduling operations efficiently but do not discuss how to achieve this goal. Some of the technological constraints of PMTs and their impact on the operation scheduling problem are discussed in Levin *et al.* [3]. They propose a procedure based on the idea of Giffler and Thompson [4] for constructing feasible semi-active schedules. Yip-Hoi and Dutta [5] present a genetic algorithm for sequencing operations.

The traditional production scheduling literature fails to address the problems that arise for PMTs due to the common assumption of serial operations. However, some simplified versions of the PMT scheduling problem are similar to problems that have been considered in the literature. These similarities will be noted in Section 2.

CAPP for PMTs opens many areas for research including feature extraction, collision avoidance, user interfaces and operation sequencing. In this paper we

[*] Corresponding author

analyze the operation sequencing problem. The specific operation sequencing problems that arise for PMTs are presented in Section 2. Section 3 provides solution methodologies for these problems. Section 4 presents computational results. Conclusions are discussed in Section 5.

2. Scheduling problem definition

The scheduling problem involves determining an operation sequence that minimizes the processing time for a given job. A job consists of one or more workpieces, each of which must have a number of operations performed on it. We are given the times required for each operation, the mode of each operation (defined below), and the precedence relations associated with each operation. The goal is to determine the sequence of operations that will minimize the overall time required to process the set of jobs. There are four problem characteristics that complicate the scheduling of operations on a PMT: (i) precedence constraints between operations; (ii) mode restrictions; (iii) assignment of operations to PMLs, and (iv) the assignment of tools to MUs. These characteristics are now described in more detail.

Precedence constraints arise for three reasons. The first involves geometric considerations. Because machining operations remove volumes of material, some operations must necessarily precede others. The second source of precedence is tolerancing. It may be necessary for one feature of a workpiece to be dimensioned off another. The third type of precedence results from manufacturing practice necessary to ensure precision.

The operations performed on a PMT may be classified into three *modes* depending on the motion of the workpiece at a PML and the motion of the MUs that are machining the workpiece. The three modes are defined as in Levin and Dutta [1]: turning – when the workpiece is rotating and the MU is stationary; milling – when the part is stationary and the MU is in motion, as in drilling or milling; contouring – when both the part and the tool are in motion, as in contour-milling. Operations that require different modes cannot be performed concurrently at the same PML due to technological limitations. The addition of mode constraints adds complexity.

Because a PMT has multiple PMLs, we must determine the set of operations that will be completed at each PML. These decisions will have a significant effect on the time required to complete the workpiece. For example, a workpiece typically visits each PML only once in order to provide a smooth material flow. We assume that parts will not make return visits to a given PML. Thus the operations assigned to a PML cannot be predecessors for operations assigned to the PMLs that the part has already visited.

The assignment of tools to MUs also has an effect on processing times. Only operations using tools on different MUs can be performed in parallel. This problem is complicated by the fact that not every PML may be accessible to every MU. Tool assignment is a difficult problem and will be explored in future research.

There may also be interactions between multiple parts in an order. Consider a part that visits two PMLs. If there is only one part in the order then the objective is to get that part off both PMLs as quickly as possible. However, if there are multiple parts in the order then when one part moves to the second PML, a new part is placed on the first PML. Now the objective is to complete the operations for both parts as quickly as possible and in a balanced fashion. Carefully sequencing these operations so that the PMT can perform some operations simultaneously will reduce the total time for an order. Thus, the objective of this sequencing problem is to minimize the total time to process the order considering a series of related problems: tool assignment, operation assignment and sequencing.

Ideally, all three problems would be solved simultaneously. As a first step toward this vision, we attack the sequencing problem resulting from known tool and operation assignments. In future work we will build on this module to address the assignment problems.

Consider the example problem data given in Table 1. An optimal operation sequence is shown in Fig. 1 and has a makespan of 45.

We now present a detailed description of scheduling problems encountered on PMTs.

Table 1. PMT example problem data

Operation	$P_{i,j}$	*Time*	*PML*	*Mode*	*MU*
1		10	1	1	2
2		7	1	3	1
3	1	9	1	2	3
4	1	6	1	2	1
5	4	8	1	3	2
6	4	6	2	2	1
7	5	3	2	1	3
8	5	8	2	1	2
9	7, 8	3	2	2	2
10	5	10	2	2	3

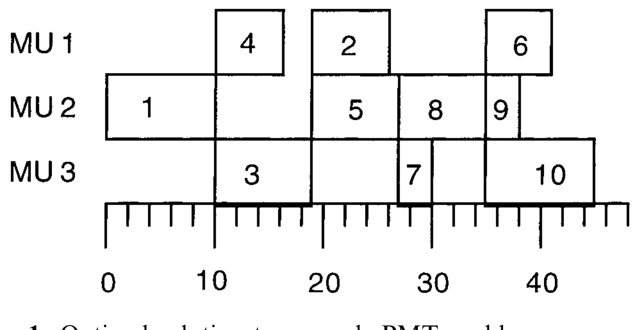

Fig. 1. Optimal solution to example PMT problem.

Problem parameters

n	=	the number of operations in the process plan, $\in \{1, 2, \ldots\}$;
i	=	the operation number, $\in \{1, 2, \ldots, n\}$;
m_i	=	the mode of operation $i, \in \{1, 2, 3\}$;
$p_{i,j}$	=	the jth immediate predecessor of operation $i, \in \{1, 2, \ldots, n\}$;
P_i	=	the set of all immediate predecessors of operation $i, \in \{1, 2, \ldots, n\}$;
t_i	=	the processing time of operation i, $\in \Re^+$;
$MU_{\max}$	=	the number of MUs in a problem, $\in \{1, 2, \ldots\}$;
$PML_{\max}$	=	the number of PMLs in a problem, $\in \{1, 2, \ldots\}$;
M	=	a large positive number;
MU_i	=	the MU for operation i, $\in \{1, 2, \ldots, MU_{\max}\}$;
PML_i	=	the PML for operation i, $\in \{1, 2, \ldots, PML_{\max}\}$.

Problem decision variables

C_i	=	the completion time of operation i, $\in \Re^+$;
$C_{\max}$	=	the maximum completion time over all n operations, $\in \Re^+$.

Additional notation

$i \prec j$		denotes that operation i must complete before operation j begins;
$i < j$		in the final schedule, operation i completes before operation j begins;
x_{ij}	=	is an indicator variable that is 1 if $i < j$.

Assumptions

We make six assumptions; (i) the MUs are continuously available; (ii) there is no preemption of operations; (iii) processing times are known in advance; (iv) operations have been assigned to PMLs; (v) tools have been assigned to MUs and there is no duplicate tooling; and (vi) the objective is to minimize the makespan (i.e., complete the job as soon as possible).

Assuming the MUs are continuously available eliminates the need to consider MU breakdowns. The solution procedures given in Section 3 can be modified to account for these breakdowns. However, the data sets tested did not contain MU breakdown information. Disallowing pre-emptions reflects the technological constraints of many metal-cutting operations. Assuming known processing times for each operation eliminates any processing time interactions that may exist due to shared machining parameters between simultaneously scheduled operations. Assuming that tools are already assigned to turrets and that there are no duplicate tools implies that each operation must be completed by a specific MU. Assumptions (iii), (iv), and (v) will be relaxed in future work. Assumptions (iv) and (v) define part of a hierarchical design problem. Future work will use the algorithms presented in this paper to build decision-support systems for the full problem.

The makespan objective (assumption (vi)) is reasonable for the PMT scheduling problem as it occurs entirely within a single machine. We assume that inventory and due dates are considered at a higher decision-making level when order-release is determined. Our objective in determining a process plan for the PMT is to get the workpieces contained within an order through as quickly as possible. Makespan is an appropriate measure for this objective.

If there are no mode conflicts, we obtain a simplified problem denoted by P1, which is a generalization of the job shop scheduling problem which is known to be NP-hard [6]. It has the same structure as a Resource-Constrained Project Scheduling Problem (RCPSP). Each operation in P1 corresponds to an activity in a RCPSP that requires one unit of a renewable resource with total availability of one unit. MUs in P1 correspond to resources in the RCPSP problem. The precedence constraints are common to both problems. The RCPSP has been studied extensively over the last 30 years [7–10]. We will build on some of these approaches to develop algorithms for more complicated variations of the problem.

Although P1 is equivalent to problems that have been previously studied in the literature, this is not true for more complicated PMT problem variations. Introducing mode constraints into the problem results in a formulation that, to the best of our knowledge, has only been previously addressed in Levin *et al.* [3] and Yip-Hoi and Dutta [5]. This problem can be formulated as the following mathematical program.

Math Program P2

Min $C_{\max}$

$$C_{\max} \geq C_i \quad \forall i = 1, 2, \ldots, n, \tag{1}$$

$$C_i \geq C_{p_{i,j}} + t_i \quad \forall i = 1, 2, \ldots, n \text{ and } \forall p_{i,j} \in P_i, \tag{2}$$

$$C_i \geq t_i + C_j + M(-x_{ij}) \quad \forall i, j = 1, 2, \ldots, n \ni MU_i = MU_j, \; i \not\prec j \text{ and } j \not\prec i, \tag{3}$$

$$C_j \geq t_j + C_i + M(x_{ij} - 1) \quad \forall i, j = 1, 2, \ldots, n \ni MU_i = MU_j, \; i \not\prec j \text{ and } j \not\prec i, \tag{4}$$

$$C_i \geq 0 \quad \forall i = 1, 2, \ldots, n, \tag{5}$$

$$C_i \geq t_i + C_j + M(-x_{ij}) \quad \forall i, j = 1, 2, \ldots, n \ni MU_i \neq MU_j, \; i \not\prec j, \; j \not\prec i, m_i \neq m_j, \text{ and } PML_i = PML_j, \tag{6}$$

$$C_j \geq t_j + C_i + M(x_{ij} - 1) \quad \forall i, j = 1, 2, \ldots, n \ni MU_i \neq MU_j, \; i \not\prec j, \; j \not\prec i, \; m_i \neq m_j, \text{ and } PML_i = PML_j, \tag{7}$$

$$x_{ij} = \begin{cases} 1 & \text{if } i < j\ , 0 \\ 0 & \text{otherwise.} \end{cases}$$

Constraint (2) ensures that each operation does not begin prior to the completion of its predecessors. The disjunctive constraints (3) and (4) ensure that each MU performs at most one operation at any given time. These disjunctive constraints are similar to those in the formulation of the job shop scheduling problem [11]. The disjunctive constraints (6) and (7) insure that only operations with the same machine mode can be scheduled concurrently. These constraints distinguish P1 from P2.

We now propose heuristic solution approaches to P2.

3. Heuristic solution procedure methodologies

We limit the solution approaches we propose for this problem to heuristic methods for two reasons. First, the addition of constraints (6) and (7) significantly increases the number of (0, 1) variables in P2 relative to P1. The problems we tested had several hundred to thousands of 0–1 variables and constraints. The large number of 0–1 variables made a direct math programming solution approach inefficient. Experimental results indicate that problem instances with 25 operations require 40 hours of computation time to solve using CPLEX on a Sun Sparc 20. Problem instances with 50 operations could not be solved in 200 hours. Second, heuristic rules are developed for problem P2 that can be extended, in future work, to include the additional complexities that result from relaxing assumptions (iii), (iv), and (v). We investigate two types of heuristic procedures. The first uses priority dispatching rules that are modified to account for the problem structure of P2. The second is a genetic algorithm that uses the random keys encoding.

3.1. *Priority rule heuristic*

Priority dispatching rules have been applied to a number of different scheduling problems [12], including problem P1. Because P2 is similar to P1, 16 priority rules from Olaguíbel and Goerlich [13] that have worked well on P1 were selected for testing on P2. The rules tested included Shortest Processing Time, Longest Processing Time, Most Immediate Successors, Least Immediate Successors, Most Total Successors, Least Total Successors, Longest Path Following, Greatest Positional Weight, Least Positional Weight, Greatest Proportional Positional Weight, Earliest Finish Time, Earliest Start Time, Latest Start Time, Latest Finish Time, Minimum Slack Time, and Least Float per Successor. For details and definitions see Olaguíbel and Goerlich [13].

For a given problem instance, each priority rule is used to construct a schedule and the best of the 16 schedules is reported. The 16 priority rules were tested on P2 type problems and the results are discussed in Section 5. In general, the dispatching rules performed poorly, which is not surprising since they do not consider the concept of modes. To improve performance, these rules were modified to account for the problem structure of P2. In P2 only operations that have the same mode can be scheduled in parallel. This was accomplished using a *delay amount* similar to that used for job shop scheduling in Norman [14].

A key factor in the priority list algorithm is the determination of feasibility of operation placement given the three different machine modes. For example, suppose that a part is being turned during time 10 through 26. Another turning operation requires a tool that is not available until time 20 and requires 10 time units. Scheduling this operation next will extend the time dedicated to turning. Whether or not we should schedule the new operation is a function of its criticality relative to others not yet scheduled. The *delay factor* is a measure of this criticality. A delay factor of five would mean that we would schedule the drilling operation if it delayed the mode change by five units or less. Hence, a higher delay factor makes an operation more critical. An appropriate delay factor for each operation is set by searching the set of possible factors.

The following algorithm describes how rules 1 to 16 were modified to incorporate the concept of *delay amounts*. Let J represent the set of schedulable operations (unscheduled operations with no unscheduled predecessors), K the set of operations that are both unscheduled and unschedulable, and ρ_i the priority rule value for operation i.

```
Step 1. Initialization. Select priority rule and calculate $\rho_i$
        values (only done once per priority rule). Initialize $J$ and $K$.
Step 2. Schedule the first operation.
            $o^* = \operatorname{Argmax}_{i \in J}\{\rho_i\}$, current mode $= m_{o^*}$,
        next mode change $= C_{o^*}$, update $J$ and $K$ to reflect scheduling $o^*$.
Step 3. Schedule subsequent operations.
        3.1
            $o^* = \operatorname{Argmax}_{i \in J}\{\rho_i\}$
        3.2 Determine if $o^*$ will be the next operation scheduled.
            3.2.1 If $m_{o^*} =$ current mode, schedule $o^*$.
            3.2.2 Else, search $J$, based on the $\rho_i$ values,
                  for operations with $m_i =$ current mode.
                  If there exists an operation, $i$, such that
                  $ES_i + t_i \leq$ delay amount + next mode
                  change then set $o^* = i$
        3.3 Schedule $o^*$ and update $J$ and $K$.
        3.4 If $J \neq \emptyset$, return to Step 3.
```

Reconsider the example problem data given in Table 1. If the Shortest Processing Time (SPT) rule is applied then the partial schedule after two operations contains operations 1 and 4. Now, applying straight SPT results in scheduling operation 2 next and the total time required at

PML 1 would be 41. However, with sufficiently large delay, operation 3 would be scheduled simultaneously with operation 4. Then operations 2 and 5 could be scheduled simultaneously and the resulting sequence would only require 27 time units at PML 1.

As the priority values and resulting schedules are easy to calculate, it is possible to check all 100 values of the different delay amounts for a given priority rule in only a few seconds of CPU time.

3.2. *A genetic algorithm*

A genetic algorithm, referred to as algorithm RKGA, is proposed that utilizes the random keys encoding introduced in Bean [15] and applied to general scheduling problems in Norman and Bean [16]. Genetic Algorithms (GAs) were introduced in Holland [17] as a method for modeling complex systems. GAs apply concepts from biological evolution to a mathematical context. The general idea is to start with randomly generated solutions and, implementing a "survival-of-the-fittest" strategy, evolve good solutions. See Michalewicz [18] for details on GAs.

There have been many previous efforts to apply GAs to sequencing and scheduling problems and several different problem encodings have been suggested (see Norman [14] for details). The problem and heuristic space methods in Storer *et al.* [19], and the random keys method of Norman and Bean [16] have shown substantial promise. Both approaches attack scheduling problems with GAs that use a multiple space approach, but differ in search philosophy, the spaces searched, and the schedule construction routines. Other successful applications of genetic algorithms to scheduling problems include Herrmann and Lee [20] and Herrmann *et al.* [21]. We selected the random keys encoding based on our previous success in modeling complex scheduling problems in Norman and Bean [16,22].

The random keys representation encodes a solution with *random* numbers. These values are used as sort *keys* to decode the solution. The chromosome is interpreted in the fitness evaluation routine in a way that avoids feasibility problems.

It was necessary to modify the general random keys genetic algorithm for P2. The precedence constraints between operations are enforced by maintaining a sorted list of random keys that only contains the schedulable operations. The structure of the precedence graph is used to bias the random keys. An operation early in the precedence graph would tend to have larger random key values, making it more likely to be early in the schedule. The goal of this biasing is to improve the GA's rate of convergence by guiding the GA to regions likely to map to schedules with good makespan values.

The schedule builder maps chromosomes of random keys to schedules of operations. It begins by using the sorted random keys sequence to determine the order for placing operations into the schedule. The schedule is constructed moving forward in time and making sure that no precedence, MU usage, or mode constraints are violated.

The GA for P2 explores a subset of semi-active schedules. A schedule is *semi-active* if it cannot be improved by sliding an operation to the left in the Gantt Chart. For a regular measure (e.g., makespan), an optimal schedule exists within the set of semi-active schedules (based on a proof in Baker [11] with only minor modifications).

To construct semi-active schedules we employ a *left-shift* operator. The *left-shift* concept uses a *delay factor* in a manner similar to the heuristic priority procedure described in Section 3.1. The number of *left-shifts* investigated is controlled dynamically by the GA. Each operation now has two genes – one containing the random key and one containing a *delay factor* with a real value between 0 and 1. The initial operation sequence is based on the sorted random key values. The schedule builder moves forward in time inserting operations. However, if scheduling operation i next results in a change of mode and creates idle time on MU_i, the remaining schedulable operations are examined to see if any of them have the same mode as the current mode. Any operations that satisfy these criteria are placed in the set δ. We then search δ to find any $j \in \delta$ where $ES_j + t_j \leq \mathit{delay\ factor}_j \times t_j + ES_i$ (ES_i represents the earliest time that operation i can start). If there exists an operation, j^*, that satisfies this criteria, then j^* is scheduled next instead of operation i. Thus *left-shifts* attempt to maximize the number of operations that can be performed in parallel. The subset of semi-active schedules that the GA searches still contains the set of active schedules. It can be shown that there exists a set in the chromosome space with non-zero measure that maps, using the schedule builder with *delay factors*, to an optimal schedule.

Utilizing *left-shifts* leads to an improvement in both solution quality and rate of convergence for the RKGA. The *left-shift* concept could be modified to ensure that the GA investigates only the set of active schedules, but it is very computationally intensive to explore all global *left-shifts* to verify that a given schedule is an active schedule.

Reconsider the example problem data given in Table 1. The random key values given in part (a) of Fig. 2 result in the schedule shown in Fig. 1. Using the delay factor concept the random keys shown in part (b) of Fig. 2 would also result in the the optimal schedule. In this case, after the first two operations are scheduled the partial schedule contains operations 1 and 4. Now, applying straight random keys results in scheduling operation 2 next. This results in a schedule that requires 41 time units at PML 1 and at least 59 time units overall. However, using the delay factors in conjunction with the random

(a)	Random keys	(0.71, 0.81, 0.46, 0.32, 0.56, 0.72, 0.23, 0.46, 0.28, 0.91)
(b)	Random keys	(0.71, 0.44, 0.46, 0.32, 0.56, 0.72, 0.23, 0.46, 0.28, 0.91)
	Delay factors	(0.16, 0.21, 0.51, 0.08, 0.61, 0.41, 0.77, 0.18, 0.27, 0.39).

Fig. 2. Sample random keys and delay factors for PMT example problem data.

keys results in the optimal sequence shown in Fig. 1. The RKGA could be run using only the straight random keys but the delay factor concept improves the performance of the RKGA.

4. Parallel machine tool computational results

Extensive computational testing was conducted on problem P2. Data sets were randomly generated to represent the types of parts machined on PMTs. An investigation of the types of parts that are manufactured using PMT's has been conducted by Yip-Hoi and Dutta [2] and Yip-Hoi [23]. Using this information we randomly generated 648 different problem instances. The problems varied in the problem characteristics described below. We examined each combination of these problem characteristics and generated three data sets for each combination.

- *Problem size*: problem sizes of 50, 75, and 100 operations were tested because these represent a reasonable range on the number of operations required to complete a part.
- *Mode assignment*: operations were assigned modes in a manner that reflects the operation to mode distribution found in practice [23]. Typically, there are more turning operations early in the precedence network and more milling and contouring later in the precedence network. We examined three different mode distributions representing a broad range of parts to evaluate what effect that had on the different solution methodologies. The first distribution comprised about 60% turning operations, 25% milling operations and about 15% contouring operations. For the second and third, these percentages were 40:40:20%; and 25:50:25%, respectively.
- *Number of MUs*: we examined problems with three, four, and five MUs. In each case the tools needed for each operation were randomly assigned to the MUs.
- *Precedence density*: we considered two types of precedence density. The first resulted in a relatively shallow precedence tree (the longest path in the network has only a few segments) with several operations in each level and few levels in the network. The second was a relatively deep precedence tree that had only a few operations in each level but extended for many levels.
- *Precedence structure*: we examined problems with an out-tree structure and a more general precedence structure where each operation could have up to four immediate predecessors.
- *PML type*: we assumed that there were two PMLs and examined two different distributions of operations to PMLs. The first PML type distributed the operations roughly evenly among the PMLs and the second type placed two-thirds of the operations at PML one and one-third at PML two. These two were tested to see what effect this could have on the effectiveness of the different solution procedures.

The presence of the mode constraints distinguishes this problem from most scheduling problems. With the exception of the heuristics of Levin *et al.* [3] and Yip-Hoi and Dutta [5], there are no existing solution methods with which to compare. Therefore, the solutions found by the two heuristic procedures are compared against each other, the procedures of Levin *et al.* [3] and Yip-Hoi and Dutta [5], and lower bounds.

Three lower bounds exist for P2. LB_1 represents the longest path in the precedence network. LB_1 is seldom the tightest bound, but it is simple to compute and could be useful for some precedence networks. A second simple lower bound is to consider the maximum amount of processing time assigned to a single MU. Letting $MUTotal_j$ represent the total amount of processing time for MU_j, then

$$LB_2 = \operatorname*{Max}_{j=1,2,\ldots,MU_{\max}} \{MUTotal_j\}.$$

LB_3 considers the fact that each MU has a fixed amount of time that it must operate in each of the three modes. Let β_{ijk} represent the sum of the processing times of all the operations that have a mode of type i and require MU_j at PML_k. There is a minimum amount of time α_{ik} that the PMT must spend in each mode i at each PML k, which is given by

$$\alpha_{ik} = \operatorname*{Max}_{j=1,2,\ldots,MU_{\max}} \{\beta_{ijk}\} \quad i = 1,2,3 \text{ and } k = 1,2.$$

Then

$$LB_3 = \operatorname*{Max}_{k=1,2}\{\alpha_{1k} + \alpha_{2k} + \alpha_{3k}\}.$$

The lower bound for the problem is set to the maximum of LB_1, LB_2, and LB_3.

The first heuristics tested are the priority rules described in Section 3. An initial implementation used the 16 priority rules as they are applied to resource-constrained project scheduling problems without modifications to account for the mode constraints. This approach performed quite poorly with results ranging from 70 to 120% above the lower bound. These heuristics were then extended to account for the mode constraints by modifying them to encourage scheduling operations with the same mode concurrently using the priority rules to determine which operation to sequence next. Simple *left-shifts* were permitted (this is the same as using a *delay factor* of zero) but the heuristic was principally driven by the priority rules. This method is referred to as PRIO1 and Table 2 shows that this method found solutions that were 20.7% above the lower bound on average across all the test problems. The second implementation, PRIO2, uses the priority rules in conjunction with delay factors as described in the algorithm found in Section 1. Recall that this method determines 16 schedules, each using a different priority rule, and then selects the best of the 16. This method works significantly better than PRIO1 as indicated in Table 2. Across the 648 test problems, the average deviation from the lower bound is 10.0%.

Table 2. Average per cent deviation from the lower bound for each of the heuristic methods for each combination of number of operations, number of MUs, precedence tree type (either outtree or more general), and precedence density type

Outtree	*Density*			*Operations*								
				50			*75*			*100*		
				MUs			*MUs*			*MUs*		
				3	*4*	*5*	*3*	*4*	*5*	*3*	*4*	*5*
0	1		PRIO1	15.8	23.9	32.9	10.7	25.3	34.9	13.0	25.9	29.2
0	1		PRIO2	5.5	13.2	21.0	3.9	13.2	21.0	4.8	12.3	15.3
0	1		LDB	9.4	17.6	25.1	6.8	16.5	23.6	6.1	15.0	19.2
0	1	YDGA	Min	*	*	*	*	*	*	*	*	*
0	1		Avg	*	*	*	*	*	*	*	*	*
0	1		Max	*	*	*	*	*	*	*	*	*
0	1	RKGA	Min	1.0	6.1	10.0	1.1	4.8	8.0	0.4	2.9	4.1
0	1		Avg	1.3	6.7	10.3	1.2	5.2	8.4	0.5	3.3	5.0
0	1		Max	1.5	7.3	10.8	1.3	5.5	9.3	0.6	3.7	5.7
1	1		PRIO1	11.7	23.0	27.1	8.5	18.1	23.6	7.1	17.5	23.7
1	1		PRIO2	4.5	10.9	16.4	2.5	8.0	11.9	1.6	6.9	12.4
1	1		LDB	5.4	15.9	19.5	4.0	10.0	14.6	2.3	9.4	14.4
1	1	YDGA	Min	8.3	17.5	19.9	12.1	24.5	29.6	15.4	29.1	41.5
1	1		Avg	10.3	19.9	22.8	14.0	26.9	32.3	17.1	31.6	44.2
1	1		Max	11.9	22.2	25.7	15.5	29.0	34.7	18.7	34.2	46.6
1	1	RKGA	Min	0.8	4.0	8.6	0.3	10.2	4.7	0.2	0.5	3.2
1	1		Avg	0.9	4.3	9.1	0.3	1.5	5.0	0.2	0.7	3.6
1	1		Max	1.2	4.0	9.7	0.3	1.9	5.5	0.2	1.1	4.0
0	2		PRIO1	15.7	24.4	36.8	15.5	28.2	36.3	16.7	28.1	40.2
0	2		PRIO2	6.8	12.7	23.3	5.7	13.9	21.0	7.2	15.3	22.8
0	2		LDB	11.4	17.7	30.0	8.7	17.1	24.9	9.9	16.8	26.5
0	2	YDGA	Min	*	*	*	*	*	*	*	*	*
0	2		Avg	*	*	*	*	*	*	*	*	*
0	2		Max	*	*	*	*	*	*	*	*	*
0	2	RKGA	Min	0.8	4.6	13.2	0.4	3.9	8.7	0.2	3.4	8.6
0	2		Avg	0.9	5.0	13.6	0.4	4.2	9.1	0.4	3.7	9.2
0	2		Max	0.9	5.5	14.2	0.5	4.7	9.7	6.6	9.9	10.0
1	2		PRIO1	13.8	26.0	30.3	11.8	16.6	31.5	11.9	18.2	23.9
1	2		PRIO2	4.8	13.0	17.2	2.6	6.8	18.1	2.8	7.0	12.0
1	2		LDB	6.9	18.6	21.2	3.9	9.8	21.4	5.1	9.7	14.2
1	2	YDGA	Min	6.4	16.8	18.8	11.9	20.7	35.5	16.9	29.4	39.6
1	2		Avg	8.6	19.4	21.7	13.7	23.1	38.1	19.3	32.1	42.9
1	2		Max	10.4	21.8	24.2	15.3	25.3	41.1	21.3	34.5	45.6
1	2	RKGA	Min	0.8	5.1	7.5	0.2	1.4	5.1	0.0	0.4	2.9
1	2		Avg	0.9	5.3	8.0	0.3	1.7	5.9	0.0	0.6	3.5
1	2		Max	0.9	5.5	8.5	0.5	2.1	6.9	0.1	0.9	4.3

* Problems could not be run using this algorithm

PRIO2 only requires a few seconds of CPU time on a Sun Sparc 20.

The second heuristic consists of the method in Levin *et al.* [3] modified to accommodate our slightly different problem structure. This method inserts one operation at a time into the schedule. At each step of the algorithm the set of schedulable operations is analyzed to determine which operations to schedule next. Three different priority rules are used to select the next operation from the set of schedulable operations. The first rule gives top priority to the operation with the most work remaining where work remaining is defined as the operation's processing time plus the sum of the processing times of all its successors. The second rule schedules operations based on MU utilization, seeking to schedule the MU with the most remaining work. The third rule seeks to minimize mode changes. The algorithm is run using each of the three different rules and the minimum makespan schedule of the three is taken as the final schedule. The algorithm requires only a few seconds of CPU time on a Sun Sparc 20. The results for this method are displayed in the row headed LDB in Table 2. Across the 648 problems the average deviation from the lower bound is 12.8%. From these results it is apparent that LDB does not perform as well as PRIO2.

The third method tested is the simple GA of Yip-Hoi and Dutta [5]. This method uses a tree-based encoding to eliminate infeasibility in the crossover operation. Yip-Hoi and Dutta [5] discuss applications to problems with an outtree precedence structure, although it is unclear how they would modify their method for more general problems. We limit our testing of this method to problems with an outtree structure. It is difficult to determine what the best parameter settings are for their GA since they only studied a single problem instance. Therefore, a preliminary study was conducted to determine the appropriate GA parameters for their method. The population size was set to 312 and the maximum number of generations to 150 to match the settings for the RKGA. Different values for the number of reproductions and mutations were tested and values of 15 and 12, respectively, were selected. We tested their method on all test problems with an outtree precedence structure. The results are presented in Table 2 in the rows denoted YDGA. The YDGA had an average deviation of 24.2% from the lower bound across all of the outtree problems. The results from Table 2 indicate that YDGA performed worse than the other proposed heuristics. It also requires more computation time than any of the other methods.

The fourth heuristic tested is the RKGA, described in Section 3.2. A description of the algorithm parameters and their values are given in the Appendix. Each problem was solved five times using a different initial random number seed resulting in five replications for each problem. Table 2 shows the minimum, average, and maximum over the five replications. Across all of the problems the average deviation from the lower bound was 3.2%. The average computation times for the 50, 75, and 100 operation problems are about 55, 85, and 125 seconds respectively on a Sun Sparc 20.

From these results it is clear that the RKGA provides the best average solution results. In addition, the RKGA's performance was very stable across the replications. The maximum deviation in solution quality from the best replication to the worst was less than 2%. The data in Table 2 also indicate that, even if we consider the worst replication, the RKGA still performs significantly better than the other heuristics with respect to solution quality. Overall, the worst replication provided a solution that was on average 6.5% better than PRIO2, 9.3% better than LDB, 17.2% better than PRIO1, and 23.8% better than the best replication of the YDGA.

We also examined how different problem characteristics affected the behavior of the different heuristics. The most significant factor was the number of MUs. As the number of MUs increased from three to five the deviation from the lower bound increased for all methods tested. This is due to two reasons. First, as the number of MUs increases it is likely that the lower bound becomes less tight. Thus, the solutions found by all approaches are probably closer to the optimal value than may appear. Second, as the number of MUs increases the problem becomes more difficult to solve because it is more difficult to optimize the number of operations being performed in parallel. The data in Table 2 show that the performance of the RKGA relative to the the other methods improved as the number of MUs increased. Figure 3 summarizes this result, showing an average across all problems for the different MU quantities (note that the YDGA data only applies to the problems with an outtree precedence structure). Table 2 also demonstrates that even for the problems with five MUs the RKGA still shows good consistency across the different replications.

The data in Table 2 indicate that there was only a minor change in performance over problem sizes for all methods except the YDGA. For the larger problems the RKGA found solutions that were closer to the value of the lower bound. However, the relative effectiveness of RKGA compared to the other solution methods was unchanged as it continued to generate solutions that were 5 to 30% better than the other methods. The performance of YDGA deteriorated as the number of operations increased, and for the large problems with more MUs it performed significantly worse than the other heuristics. This may be an indication that the YDGA encoding is not well suited to problems with several MUs.

The final problem characteristic that affected the heuristics' performance was the precedence structure. The data in Table 2 indicate that the outtree precedence structure (denoted as outtree = 1 in Table 2) in general led to problems that were easier to solve. The solutions

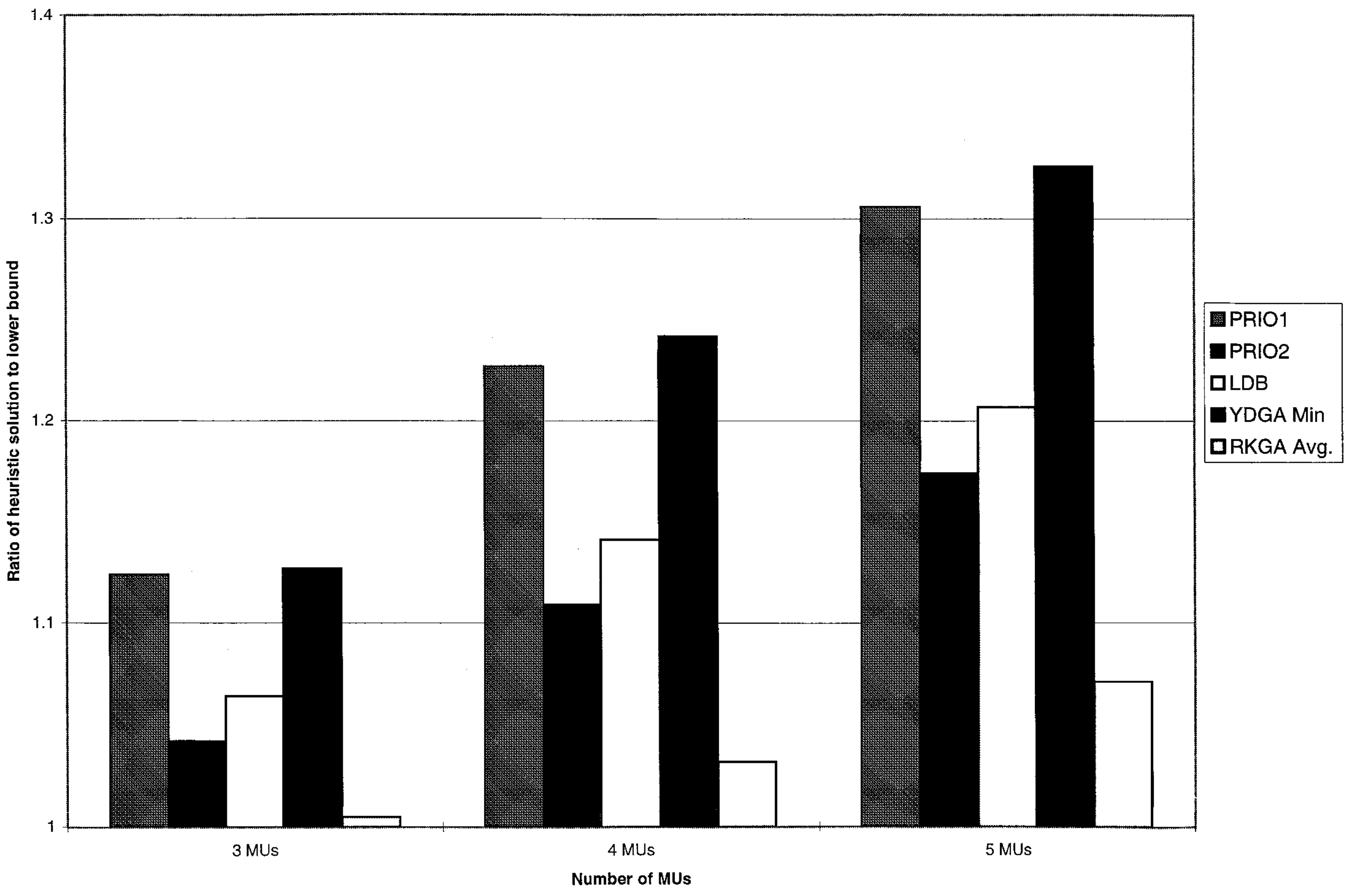

Fig. 3. Summary of the average percent deviation from the lower bound for each heuristic method for all of the problems with three, four and five MUs

found by PRIO1, PRIO2, and LDB had on average about five points greater error for the problems with the more general precedence structure (denoted as outtree = 0 in Table 2) than for those with the outtree structure. Similarly, the RKGA solutions had about two points greater error for the problems with the more general structure. The difference in performance may be due to the fact that the lower bounds are looser for the more general precedence structure. It is interesting to note that the performance of RKGA was more robust to the precedence structure than the other heuristics. YDGA is not discussed because it was only applied to problems with an outtree precedence structure. The other factors of mode assignment, precedence density, and PML type did not have a significant affect on the performance of the heuristics.

Overall, for P2 the proposed heuristics found good solutions that were better than those found by existing methods. The priority rule heuristic found solutions that were on average about 10% above the lower bound. The RKGA found solutions that averaged 3.2% above the lower bound. Both of these represent significant improvements over simple dispatching rules and the LDB and YDGA heuristics.

5. Conclusions and further research

In this paper we introduce some unique scheduling problems that arise for PMTs. The capability of these machines to perform simultaneous operations on single and multiple parts creates constraints not found in traditional scheduling. Two heuristics are introduced for these problems. The first uses priority dispatching rules and a delay factor concept. The second uses a genetic algorithm based on the random keys encoding. Computational tests are presented for 648 data sets. The priority rule-based heuristic finds reasonably good solutions with an average deviation from the lower bound of less than 10% and improves on Levin *et al.* [3] and also Yip-Hoi and Dutta [5], the only algorithms in the literature. An advantage of the priority rule-based heuristic is that it requires little computation time. The RKGA finds better solutions than the priority rule heuristic, within 3.2% of the lower bound on average. However, this method requires more computation time than the priority rule heuristic method.

In future research, we will explore several problem extensions to P2. The first extension relaxes the assumption that operations have already been assigned to PMLs. This creates another level of decisions but only requires minor changes in the RKGA model. The second is relaxing the assumption that processing times are deterministic. Because operations performed in parallel share machining parameters, the processing times of operations will depend on the sequence of the operations. The third extension will be to include the assignment of tooling to turrets in the model.

Acknowledgments

The authors would like to thank two anonymous referees for their helpful suggestions and Derek Yip-Hoi for providing a copy of his genetic algorithm code.

References

[1] Levin, J. and Dutta, D. (1992) Computer-aided process planning for parallel machines. *Journal of Manufacturing Systems*, **11**, 79–92.

[2] Yip-Hoi, D. and Dutta, D. (1993) Issues in computer-aided process planning for parallel machines. *Advances in Design Automation*, **65**, 153–161.

[3] Levin, J., Dutta, D. and Bean, J.C. (1993) PMPS: a prototype CAPP system for parallel machining. Technical Report, University of Michigan, Ann Arbor, MI 48109-2117.

[4] Giffler, B. and Thompson, G.L. (1960) Algorithms for solving production scheduling problems. *Operations Research*, **8**, 487–503.

[5] Yip-Hoi, D. and Dutta, D. (1996) A genetic algorithm application for sequencing operations in process planning for parallel machining, *IIE Transactions*, **28**, 55–68.

[6] Garey, M.R., Johnson, D.S. and Sethi, R. (1976) The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, **1**, 117–129.

[7] Bell, C.E. and Park, K. (1990) Solving resource-constrained project scheduling problem by A* search. *Naval Research Logistics*, **37**, 61–84.

[8] Demeulemeester, E. and Herroelen, W. (1992) Branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, **38**, 1803–1818.

[9] Talbot, F.B. and Patterson, J.H. (1978) An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. *Management Science*, **24**, 1163–1174.

[10] Bell, C.E. and Han, H. (1991) A new heuristic method in resource-constrained project scheduling. *Naval Research Logistics*, **38**, 315–331.

[11] Baker, K. (1974) *Introduction to Sequencing and Scheduling*, Wiley, New York.

[12] Morton, T.E. and Pentico, D.W. (1993) *Heuristic Scheduling Systems*, Wiley, New York.

[13] Olaguíbel, R.A. and Goerlich, J.M.T. (1989) Heuristic algorithms For resource-constrained project scheduling: a review and an empirical analysis, in *Advances in Project Scheduling*, Slowiński, R. and Weglarz, J. (eds), Elsevier, pp. 113–134.

[14] Norman, B.A. (1995) Scheduling using the random keys genetic algorithm. Unpublished PhD. dissertation, University of Michigan, Ann Arbor, 48109-2117.

[15] Bean, J.C. (1994) Genetics and random keys for sequencing and optimization. *ORSA Journal on Computing*, **6**, 154–160.

[16] Norman, B.A. and Bean, J.C. (1999) Random keys genetic algorithm for complex scheduling problems. *Naval Research Logistics*, **46**, 199–211.

[17] Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI.

[18] Michalewicz, Z. (1994) *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd edn, Springer-Verlag, New York.

[19] Storer, R., Wu, S. and Vaccari, R. (1992) New search spaces for sequencing problems with application to job shop scheduling. *Management Sciences*, **38**, 1495–1509.

[20] Herrmann, J.W. and Lee, C.-Y. (1995) Solving a class scheduling problem with a genetic algorithm. *ORSA Journal on Computing*, **7**, 443–452.

[21] Herrmann, J.W., Lee, C.-Y. and Hinchman, J. (1995) Global job shop scheduling with a genetic algorithm. *Production and Operations Management*, **4**, 30–45.

[22] Norman, B. and Bean, J. (1996) Random keys genetic algorithm for job shop scheduling. *Engineering Design and Automation*, **3**, 145–156.

[23] Yip-Hoi, D. (1994) Personal communication.

Appendix

The following parameter values were used based on preliminary testing. See Norman [14] for a complete discussion of these parameters.

- Population size = 312.
- Maximum run length = 150 generations.
- Clones = 15. This clones or copies the 15 best solutions from generation i directly into generation $i+1$. (This is often referred to as an elitest strategy.)
- Immigrants = 36. Immigration is a type of mutation where randomly generated chromosomes are introduced into the population each generation. This introduces new genetic material into the population when the immigrants are chosen as parents in the crossover procedure.
- Mutation chromosomes = 54. This is a more conventional form of mutation. Since the random keys and delay factors are real values we mutate them by perturbing the current value by a random variate uniformly distributed between -0.5 and 0.5. Each gene is mutated with a probability of 0.50.
- Crossover probability = 0.7. Uniform crossover is utilized and the probability of selecting the allele value from parent 1 is 0.7.
- Tournament selection with $t = 2$. To select each parent for the crossover procedure two solutions are randomly chosen from the current population and the best of the two is retained as the parent.

Biographies

Bryan A. Norman, is an Assistant Professor of Industrial Engineering at the University of Pittsburgh. He received a Ph.D. degree in Industrial Engineering from the University of Michigan in 1995, where he

was a National Science Foundation Fellowship holder, and has a Master's Degree and a B.S. in Industrial Engineering from the University of Oklahoma. His research interests include the optimization of complex problems in manufacturing systems with a specialty in heuristic optimization. His areas of application include scheduling, sequencing, job rotation, assembly line balancing and facility layout. His research has been funded by the National Science Foundation, the Ben Franklin Technology Center of Western Pennsylvania, the Society of Manufacturing Engineers and by local industry. He is a senior member of the Institute of Industrial Engineers.

James C. Bean is a Professor in the Department of Industrial and Operations Engineering at the University of Michigan and Ford Motor Company Co-Director of the Tauber Manufacturing Institute, a University-wide program for manufacturing education and research. He has earned a Master's Degree and a Ph.D. from Stanford University in Operations Research and a B.S. in Mathematics from Harvey Mudd College. Professor Bean's research interests are in genetic algorithms, integer programing and Markov decision processes as applied to equipment replacement, capacity expansion, production and scheduling. He has published in *Mathematics of Operations Research*, *Operations Research*, *Mathematical Programming*, *Management Science*, *Naval Research Logistics*, *IIE Transactions*, *The Engineering Economist*, *Interfaces*, *Information Processing and Management* and the *Journal of Biomechanics*. Professor Bean is an Associate Editor for the *Journal of Scheduling*, past Associate Editor of *Management Science* and Past-Editor of the *ORSA/TIMS Annual Comprehensive Index*. Professor Bean is Secretary and former Vice-President for Information Technology of the Institute for Operations Research and the Management Sciences (INFORMS). Professor Bean has received seven outstanding teaching awards. Professor Bean has worked on various industrial projects with companies such as Penford Products (scheduling), Homart Development (divestiture scheduling), General Motors (scheduling and equipment replacement/capacity planning). Michigan Consolidated Gas Company and IBM (equipment replacement), Bethlehem Steel (capacity planning) and Tektronix (forecasting).