OPERATION SCHEDULING FOR PARALLEL MACHINE TOOLS

Bryan A. Norman*
Department of Industrial Engineering
University of Pittsburgh
Pittsburgh, PA 15261

James C. Bean
Department of Industrial & Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

Technical Report 95-29

November 1995

*Corresponding author

The Importance of Scheduling Operations on Parallel Machine Tools

We discuss the important problem of sequencing operations on a parallel machine tool. Parallel machine tools are CNC machines that contain multiple spindles and multiple tooling heads. Recent research has explored the development of automated process planning systems for these machines. These process planning systems perform many tasks including feature identification and extraction, selection of manufacturing method, determination of operation sequence, and provide CNC programs to produce the parts. While previously published research has explored other aspects of the process planning problem for these machines, little work has been done investigating the operations sequencing problem. However, selecting the proper operations sequence can have a significant impact on the time required to produce a given part and therefore affect the overall effectiveness of the process planning system.

The operation sequencing problems that arise on parallel machine tools are different than those found in the traditional sequencing and scheduling literature. The principal difference it that there exists an opportunity to perform multiple operations simultaneously on a single part. But the situation is complicated by the fact that only certain types of operations may be performed simultaneously. For example, you cannot perform a turning operation and a radial drilling operation at the same time. There is also the problem of determining the order in which the turrets should visit the different spindle locations. We formulate these sequencing problems and present two methodologies for solving them. The first approach uses priority dispatching rules that are modified to reflect the unique problem characteristics that arise for parallel machine tools. The second approach utilizes a genetic algorithm.

Abstract

We introduce unique scheduling problems that arise for multiple spindle machine tools. The capability of these machines to perform simultaneous operations on one or more parts creates constraints that are not found in the traditional scheduling literature. Two types of solution methodologies are introduced for these problems. The first utilizes priority dispatching rules and a delay factor concept to enhance the quality of the developed solutions. The second utilizes 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. In order to gain the full benefit of improvements in hardware, CAPP software must be developed that can exploit these improvements. Parallel Machine Tools represent a relatively new development. A key difference between Parallel Machine Tools and conventional CNC machines is that the former contains multiple spindles and can hold multiple workpieces concurrently. A typical Parallel Machine Tool has one main spindle and a number of subspindles that may not be identical to the main spindle.

To properly describe Parallel Machine Tools it is necessary to define some terms. We retain the terminology introduced in [16]. A *part machining location* (PML) refers to one valid workholding location. The main spindle and subspindle(s) always represent valid PMLs. However, with special fixturing, a turret on a machine tool can also be a valid PML. The maximum number of parts that can be placed on a Parallel Machine Tool at one time is not greater than the number of PMLs. A *machining unit* (MU) refers to a tool holding device. Tool holding devices may hold a single tool or a turret containing multiple tools. Relative motions between the machine tool on the MU and the workpiece held in the PML accomplish the machining. Conventional machines have only one MU and one PML. *Parallel Machine Tools* (PMTs) have $N(\geq 1)$ PMLs and $M(\geq 1)$ MUs.

CAPP for PMTs opens many areas for research including feature extraction, collision avoidance, user interfaces and operation sequencing. In this paper we analyze the operation sequencing problem. Section 2 reviews the relevant literature concerning Parallel Machine Tools. The specific operation sequencing problems that arise for Parallel Machine Tools are presented in Section 3. Section 4 provides solution methodologies for these problems. Section 5 presents computational results. Conclusions and future research are discussed in Section 6.

2 Literature Review

Due to the presence of multiple PMLs and multiple MUs, Parallel Machine Tools offer new challenges for operations control and 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 any given time. However, Parallel Machine Tools are not restricted to these assumptions.

The scheduling of operations on a Parallel Machine Tool has not received much attention in the literature. Two papers that discuss process planning for Parallel Machine Tools, [16] and [30], mention the importance of scheduling operations efficiently but do not discuss specific methodologies for achieving this goal. Some of the technological constraints of Parallel Machine Tools and their impact on the operation scheduling problem are discussed in [17].

They propose a procedure based on the idea of [12] for constructing feasible semi-active schedules.

The traditional production scheduling and FMS literature fails to address the scheduling problems that arise for Parallel Machine Tools due to the common assumption of serial operations. However, some simplified versions of the Parallel Machine Tool scheduling problem are similar to problems that have been considered in the literature. These similarities will be noted in Section 3.

3 Scheduling Problem Definition

Scheduling operations on a PMT is complicated by four characteristics of the problem: precedence constraints between the operations, mode restrictions, determination of PML for each operation, and the assignment of tools to MUs. Each of these 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 other operations. For example, a hole must be drilled before it can be reamed and finished. The second source of precedence is tolerancing. It may be necessary for one feature of a workpiece to be dimensioned off of another. For example, the location of a hole may be determined relative to a finished face of the workpiece. In this case, it is necessary to finish the face of the workpiece before drilling the hole to insure that the hole is properly located. The third type of precedence results from manufacturing practice. Consider a part which has a hole on a sloped face of a part. Manufacturing practice dictates that the hole be drilled prior to the workpiece face being machined to a slope. Otherwise, the sloped face results in increased drill bit slippage.

The operations performed on a PMT may be classified into three *modes* or categories 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 [16]: 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. For example, it is not possible to perform a turning operation and a radial drilling operation (where the drill bit enters the workpiece perpendicular to the axis of rotation) in parallel. The addition of mode constraints results in additional complexities. The problem formulations presented later in this section show how the number of disjunctive constraints increases when mode constraints are included in a problem.

Because a PMT has multiple PMLs, it is necessary to determine the set of operations that will be completed at each PML. Determining which operations to perform at each PML will have a significant effect on the time required to complete the workpiece. For example, a workpiece typically visits each PML only one time in order to provide a smooth material flow. While parts could make return visits to a given PML, we assume this is not the case for the remainder of this discussion.

The assignment of tools to MUs also has a significant effect upon the processing time for a part. Only operations that have tools on different MUs can be performed in parallel since only one of the tools (assuming a turret is used) on a MU can be accessed at any point in time. The tool assignment problem alone is a rather difficult problem and will be explored in future research.

We now present a detailed description of scheduling problems encountered on PMTs. We begin by defining problem parameters, decision variables, and notation that are used in the problem formulations and discussion. We also discuss assumptions that are common to all of the problems investigated.

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, $\in \Re^+$

Additional Notation

$i \prec j$	denotes that operation i must precede operation j but operation i is not necessarily the immediate predecessor of operation j.
$i < j$	in the final schedule, operation i completes before operation j begins (i and j may not have a precedence relationship)

Assumptions

We list six assumptions concerning the problem discussed in this paper.

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 are previously assigned to MUs and there is no duplicate tooling.

VI. The objective is to minimize the makespan.

Assuming the MUs are continuously available eliminates the need to consider MU breakdowns. The solution procedures given in Section 4 can be modified to account for these breakdowns. However, the data sets tested did not contain MU breakdown information. Not permitting preemptions reflects the technological constraints of many metal cutting operations. Once an operation is started it should proceed to its completion in order to produce a better finish and to reduce tool wear. 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. The makespan objective is reasonable for the PMT scheduling problem since it has few inventory or due time implications. In the PMT scheduling problem, the objective is to get parts off of the machine as quickly as possible in order to make the machine available for other parts. This corresponds to minimizing the makespan for the process plan.

If we also assume that there are no mode conflicts, we obtain a simplified problem denoted P(1). This problem is a generalization of the job shop scheduling problem which has been demonstrated to be NP-hard [11]. It has the same structure as a resource-constrained project scheduling problem (RCPSP). Each operation in P(1) corresponds to an activity in a RCPSP that requires one unit of a renewable resource with total availability of one unit. MUs in P(1) correspond to resources in the RCPSP problem. The precedence constraints are common to both problems. While the problem structure is the same, the time horizon for many RCPSP applications is quite different than for P(1). The RCPSP is often found in project scheduling contexts where activity durations may be hours, days, or weeks while the activity times are usually seconds for the PMT operation scheduling problem. However, the RCPSP can also be applied to job shop settings where the activity times would be of the same order as those found in P(1).

The RCPSP has been studied extensively over the last 30 years. Optimal solution procedures for the RCPSP have been proposed by [3], [6], [9], and [26]. All of these procedures utilize a branch-and-bound solution methodology. The approaches differ in their choice of branching rules and in their methods for determining lower bounds. Due to the computational requirements of these branch-and-bound algorithms, none of them have been utilized to solve problems with more than 50 activities. Most of the problems tested contain 30 activities or less. Heuristics have been proposed by [5], [10], [15], [22], [28], [4], [7], and [27]. All of these efforts are based on determining priority rules to choose which activity should be scheduled next subject to the precedence constraints. Many of these methods utilize calculations from the Critical Path Method (CPM). We will build on some of these heuristics to develop approaches for more complicated variations of the problem.

Although P(1) 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 problem that, to the best of our knowledge, has only been previously addressed in [17]. This problem can be formulated as the following mathematical program:

Math Program P(2)

Min C_{max}

$C_{max} \geq C_i$	$\forall i = 1, 2, \ldots, n$	(1.1)
$C_i \geq C_{p_{i,j}} + t_i$	$\forall i = 1, 2, \ldots, n$ and $\forall\, p_{i,j} \in P_i$	(1.2)
$C_i \geq t_i + C_j + M\,(-x_{ij})$	$\forall i, j = 1, 2, \ldots, n \ni \mathrm{MU}_i = \mathrm{MU}_j, i \not\prec j$ and $j \not\prec i$ and $j \not\prec i$	(1.3)
$C_j \geq t_j + C_i + M\,(x_{ij} - 1)$	$\forall i, j = 1, 2, \ldots, n \ni \mathrm{MU}_i = \mathrm{MU}_j, i \not\prec j$ and $j \not\prec i$ and $j \not\prec i$	(1.4)
$C_i \geq 0$	$\forall i = 1, 2, \ldots, n$	(1.5)
$C_i \geq t_i + C_j + M\,(-y_{ij})$	$\forall i, j = 1, 2, \ldots, n \ni \mathrm{MU}_i \neq \mathrm{MU}_j, i \not\prec j,$ $j \not\prec i, m_i \neq m_j,$ and $\mathrm{PML}_i = \mathrm{PML}_j$	(2.1)
$C_j \geq t_j + C_i + M\,(y_{ij} - 1)$	$\forall i, j = 1, 2, \ldots, n \ni \mathrm{MU}_i \neq \mathrm{MU}_j, i \not\prec j,$ $j \not\prec i, m_i \neq m_j,$ and $\mathrm{PML}_i = \mathrm{PML}_j$	(2.2)

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

Constraint (1.2) ensures that each operation does not begin prior to the completion of its predecessors. The disjunctive constraints (1.3) and (1.4) insure that each MU performs at most one operation at any given time. These disjunctive constraints are similar to those presented in the formulation of the job shop scheduling problem (see [1]).

The disjunctive constraints (2.1) and (2.2) insure that only operations with the same machine mode can be scheduled concurrently. These constraints separate P(1) and P(2).

We now propose heuristic solutions approaches to problem P(2).

4 Heuristic Solution Procedure Methodologies

We limit the solution methodologies that we propose for this problem to heuristic methods for two reasons. First, the addition of constraints (2.1) and (2.2) significantly increases the number of (0,1) variables in P(2) relative to P(1). Due to the large number of (0,1) variables that would be present, even in a problem with only a few operations, we are not optimistic about the prospects of implementing a branch-and-bound algorithm for P(2). Second, heuristic rules are developed for problem P(2) 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 P(2). The second is a genetic algorithm that utilizes the random keys encoding.

4.1 Priority Rule Heuristic

Priority dispatching rules have been applied to a number of different scheduling problems (see [23] and [19]). As discussed in Section 3, several dispatching rules have been applied to problem P(1). Because problem P(2) has some similarities to problem P(1), priority dispatching rules that had worked well on problem P(1) were selected for testing on these problems. A total of 16 different priority rules from [22] were tested. For a given problem instance, each priority rule is used to construct a schedule and the final schedule is the best of the 16. To explain the different priority rules it is necessary to define some additional notation:

S_i	The set of all operations that are successors of operation i.
PW_i	Positional Weight of operation i, $PW_i = \sum_{k \in S_i} t_k$.

The 16 priority rules shown in Table 1 were tested on P(2) type problems and the results are shown in Table 3. The poor performance is not surprising since these rules do not consider the concept of modes. To improve performance, these rules were modified to account for the problem structures found in P(2). In P(2) only operations that have the same mode can be scheduled in parallel so it is desirable to create a schedule that exploits this fact. This was accomplished using a *delay amount* similar to the one used for job shop scheduling in [20]. The underlying idea is to shift operations in the schedule such that operations with the same mode are scheduled in parallel.

1. Shortest Processing Time (SPT)
2. Longest Processing Time (LPT)
3. Most Immediate Successors (MIS)
4. Least Immediate Successors (LIS)
5. Most Total Successors (MTS)
6. Least Total Successors (LTS)
7. Longest Path Following (LPF)
8. Greatest Positional Weight (GPW)
9. Least Positional Weight (LPW)
10. Greatest Proportional Positional Weight (GPPW), $GPPW_i = \frac{t_i + PW_i}{t_i}$
11. First Come First Serve (FCFS)
12. Earliest Finish Time (EFT), $EFT_i = EFT_{p_i} + t_i$.
13. Latest Start Time (LST), $LST_i = \underset{j \in S_i}{Max}\{LST_j - t_i\}$ for all i such that$S_i \neq \emptyset$, else $LST_i = \underset{all\ i}{Max}\{LFT_i\}$ for all i such that $S_i = \emptyset$.
14. Latest Finish Time (LFT), $LFT_i = LST_i + t_i$.
15. Minimum Slack Time (MST), $MST_i = LST_i - EST_i$, where $EST_i = EFT_i - t_i$.
16. Least Float per Successor (LFS), $LFS_i = \frac{LST_i - EST_i}{|S_i|}$

Table 1: Priority Rules

Two methods are used to determine how much to delay an operation, one using integer delay factors and one using real valued delay factors. The first method sets the *delay amount* equal to an integer ranging from 1 to 100. This method is denoted DELINT. A range of 1 to 100 was selected because 100 represents the largest possible processing time for an operation in the test data sets. The second method uses a *delay factor* that is a real number from the set {0.01, 0.02, 0.03 ..., 1.00}. This method is denoted DELREAL. The *delay factor* is multiplied by an operation's processing time to determine the *delay amount.* For both methods, if $ES_i + t_i \leq$ *delay amount* + *next mode change* then operation i precedes operation 4. Thus, while operation 4 has the best priority value it is not necessarily the next operation scheduled.

The algorithm in Table 2 describes how rules 1 to 16 were modified to incorporate the concept of *delay amounts.* Let J represent the set of schedulable operations (those with no unscheduled predecessors) and K the set of unschedulable operations. The first step is to select a priority rule, calculate the priority value, ρ_i, for each operation I, and initialize J and K. The second step is to schedule the first operation. Select the operation, o^*, in J that has the maximum ρ value and schedule it. Set *current mode*, a variable indicating the mode at the current point in time of the schedule builder, to the mode of operation o^*. Set *next mode change time* to the completion time of o^*. Find all the operations in K that became schedulable due to scheduling o^* and move them from K to J. The third step is repeated until all the operations are scheduled. Step 3.1 selects the operation, o^*, in J with the maximum ρ value. If the mode of o^* equals the *current mode*, schedule o^*. Otherwise, search J, based on the ρ_i values, for operations with the same mode as *current mode.* If there exists an operation, i, such that its earliest start time, ES_i, plus its processing time, t_i, is less than or equal to the *delay amount* plus *next mode change* then set $o^* = i$. Note that operation i does not have to require the same MU as o^* in order to consider scheduling it prior to o^*. Upon determining $o^* = i$, find all the operations in K that became schedulable due to scheduling o^* and remove them from K and place them in J. If $J \neq \emptyset$, repeat step 3.

```
1. **Initialization.** Select priority rule and calculate $\rho_i$ values. Initialize $J$ and $K$.
2. **Schedule the first operation.** $o^* = \underset{i \in J}{Argmax}\{\rho_i\}$, *current mode* $= m_{o^*}$,
   *next mode change* $= C_{o^*}$, update $J$ and $K$ to reflect scheduling $o^*$.
3. **Schedule subsequent operations.**
   3.1 $o^* = \underset{i \in J}{Argmax}\{\rho_i\}$
   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 3.
```

Table 2: Algorithm for Heuristic Priority Procedure

Because the priority values and resulting schedules are fast to calculate, it is possible to check all 100 values for a given priority rule in less than 1 second of cpu time. Therefore, the entire ranges from 1 to 100 for DELINT and 0.01 to 1.00 for DELREAL are tested in the heuristic procedure.

4.2 Genetic Algorithm Solution Methodology

A genetic algorithm is proposed that utilizes the random keys encoding introduced in [2] and applied to general scheduling problems in [21]. Genetic algorithms (GAs) were introduced by [14] 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 [13] [8], or [18] for details on GAs.

There have been several previous research efforts to apply GAs to sequencing and scheduling problems and several different problem encodings have been suggested (see [20] for details). The problem and heuristic space methods of [24] and [25], and the random keys method of [21] have shown substantial promise. Both approaches attack scheduling problems (though not the same ones) via genetic algorithms that use a multiple space approach. However, the approaches differ in that the search philosophy is different, the spaces searched are different, and schedule construction routines are different. We selected the random keys encoding based on our previous success in modeling scheduling problems containing a high degree of complexity [21].

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 make modifications to the general random keys genetic algorithm in order to apply it to P(2). The precedence constraints between operations are captured by maintaining a sorted list of random keys where the sort list only contains the random keys for the operations that are currently schedulable based on the precedence relationships. Additionally, the structure of the precedence graph is utilized to bias the random keys in a manner similar to [20]. The goal of biasing is to improve the GA's rate of convergence by guiding the GA to regions of the chromosome space that are likely to map to schedules with good makespan values.

An important part of any GA is the mechanism for determining the fitness of different chromosomes. In the context of scheduling problems, we refer to this mechanism as a schedule builder. The schedule builder maps chromosomes of random keys to schedules of operations. The schedule builder begins by using the sorted random keys sequence to determine the order for placing operations into the schedule. The schedule is constructed in a manner that moves forward in time and insures that no precedence, MU usage, or mode constraints are violated.

Initially, the GA for P(2) utilized a schedule builder that explored the set of semi-active schedules (as defined in [1]). An optimal schedule exists within the set of semi-active based on a proof in [1] that only requires minor modifications to account for the concept of modes. However, the mode constraints in P(2) result in a large number of semi-active schedules. Therefore, the schedule builder was modified to incorporate *left-shifts* that result in the GA searching only a subset of the set of semi-active schedules.

The *left-shift* concept uses a *delay factor* in a manner similar to the heuristic priority procedure described in Table 2. 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* that has a real value between 0 and 1. The initial operation sequence is based on the sorted random key values of the operations. 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 a search is made of the remaining schedulable operations 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, based on the random key values for each operation $j \in \delta$, for any j where $ES_j + t_j \leq delayfactor_j * t_j + ES_i$. If there exists an operation, j^*, that satisfies this criteria then j^* becomes the next operation to schedule in place of operation i. In this way *left-shifts* attempt to maximize the number of operations that can be performed in parallel. The subset of the set of semi-active schedules that the GA searches using *left-shifts* still contains the set of active schedules and the following theorem can be proven.

Theorem 1 *There exists a set with nonzero measure in the chromosome space that maps, using the schedule builder that includes delay factors, to an optimal schedule.*

Proof: Similar to that of Lemma 3 and Lemma 4 in [20] with minor modifications to account for the mode constraints.

This theorem demonstrates the validity of building schedules using *left-shifts* because the GA search space still contains an optimal region. Utilizing *left-shifts* leads to an improvement in both the quality of solutions found and the rate of convergence for the RKGA. The *left-shift* concept could be modified to insure that the GA only investigates the set of active schedules but it is a very computationally intensive task to explore all global left-shifts and insure that a given schedule is an active schedule.

5 Parallel Machine Tool Computational Results

Computational testing was conducted on problem P(2). Data sets were randomly generated that represent the types of parts that are machined on parallel machine tools. An investigation of the types of parts that are manufactured using PMT's has been conducted by [30] and [29]. Using this information we randomly generated 10 different data sets for three different sizes of problems for a total of 30 data sets. 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. Operations were assigned modes in a manner that reflects the operation to mode distribution found in parts actually made on PMTs [29]. Typically, there are more turning operations early in the precedence network and more milling and contouring later in the precedence network. The tools needed for each operation were randomly assigned to the MUs.

The presence of the mode constraints makes this problem different from most scheduling problems presented in the literature. With the exception of the heuristic of [17], 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 procedure of [17], and lower bounds. Values of the best known solution are also presented for each problem instance.

Two lower bounds exist for P(2). LB_1 represents the longest path in the network. $LB_1 = \underset{all\ i}{Max}\{LFT_i\}$. LB_1 seldom represents the tightest bound but is simple to compute and could be useful for some precedence networks. LB_2 utilizes more problem structure and considers the fact that each MU has a fixed amount of time that it must operate in each of the three modes. Let β_{ij} represent the sum of the processing times of all the operations that have a mode of type i and require MU_j. There is a minimum amount of time that the PMT must spend in each mode i, call this α_i,

$$\alpha_i = \underset{j=1,2,\ldots,MU_{max}}{Max}\{\beta_{ij}\} \quad i = 1,2,3.$$

Then $LB_2 = \alpha_1 + \alpha_2 + \alpha_3$. The lower bound for the problem is set to the maximum of LB_1 and LB_2.

The first heuristic methodology tested is the method described in Section 4 that is based on priority rules. The first priority rule implementation, PRIO1, uses the 16 priority rules described in Section 4 as they are applied to resource constrained project scheduling problems - no modifications are made to account for the mode constraints. The results of Table 3 show that this method performs poorly. The second implementation, PRIO2, utilizes the priority rules in conjunction with delay factors as described in the algorithm found in Table 2. 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 as indicated in Table 3. Across the 30 test problems, the average deviation from the lower bound is 3.2% and from the best known solution is 2.2%. PRIO2 only requires a few seconds of cpu time on a Sun Sparc 20.

		Best				RKGA1			RKGA2		
Problem	LB	Known	PRIO1	PRIO2	LDB	Min.	Avg.	Max.	Min.	Avg.	Max.
50.1	1112	1164	1754	1191	1191	1164	1165	1169	1164	1166	1169
50.2	1097	1097	1743	1139	1139	1115	1122	1126	1115	1117	1124
50.3	1117	1166	1655	1179	1179	1166	1167	1174	1166	1166	1166
50.4	1078	1104	1810	1144	1223	1112	1131	1147	1121	1121	1122
50.5	1230	1230	1790	1271	1304	1230	1230	1230	1230	1230	1230
50.6	1156	1165	1686	1178	1208	1165	1166	1169	1165	1166	1178
50.7	1108	1108	1608	1122	1122	1108	1108	1108	1108	1108	1108
50.8	1089	1089	1838	1129	1215	1089	1095	1103	1093	1099	1102
50.9	1147	1149	1921	1217	1217	1149	1149	1149	1149	1150	1158
50.10	1515	1515	2122	1515	1536	1523	1531	1532	1515	1515	1515
75.1	1707	1707	2965	1803	1809	1707	1708	1721	1707	1707	1708
75.2	1670	1739	2573	1761	1841	1756	1758	1760	1754	1755	1755
75.3	1651	1654	2716	1693	1693	1680	1680	1680	1680	1680	1680
75.4	1758	1758	2944	1761	1818	1758	1759	1762	1758	1758	1758
75.5	1995	1995	2854	1995	1995	1995	1995	1995	1995	1995	1995
75.6	1567	1567	3077	1636	1706	1588	1603	1637	1567	1573	1588
75.7	1920	1994	2926	1994	2001	1994	1994	1994	1994	1994	1994
75.8	1872	1883	3191	1905	1911	1897	1935	1965	1888	1891	1891
75.9	1444	1444	2504	1463	1561	1444	1446	1452	1444	1444	1444
75.10	1934	1951	3029	1967	2041	1962	1992	2037	1955	1962	1967
100.1	2129	2157	3449	2304	2306	2179	2272	2323	2175	2217	2281
100.2	2123	2198	3650	2214	2214	2198	2201	2209	2198	2200	2214
100.3	1985	1997	3928	2073	2215	2037	2085	2103	2000	2012	2022
100.4	1745	1761	3281	1810	1902	1822	1845	1873	1761	1798	1795
100.5	2259	2259	3695	2269	2605	2259	2260	2269	2259	2259	2259
100.6	2160	2209	4043	2244	2292	2210	2224	2249	2216	2223	2235
100.7	2184	2187	3917	2292	2325	2216	2221	2232	2187	2206	2223
100.8	1992	2014	3608	2083	2083	2025	2044	2070	2024	2054	2072
100.9	2438	2438	4175	2451	2481	2438	2445	2458	2438	2445	2451
100.10	2253	2253	3523	2253	2253	2253	2253	2254	2253	2253	2253
Avg. Dev. LB			66.5	3.2	5.7	1.6	2.3	3.0	1.3	1.6	2.0
Avg. Dev. BK			64.8	2.2	4.6	0.6	1.2	2.0	0.2	0.6	1.0

Table 3: P(2) PMT Test Problem Results.

The second heuristic consists of the method in [17] modified to accommodate our slightly different problem structure. This method inserts one job 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 job from the set of schedulable operations. The first rule gives top priority to the operation with the most work remaining where work remaining equals 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 and seeks 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 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 column headed LDB in Table 3. Across the thirty test problems the average deviation from the lower bound is 5.7% and from the best know solution is 4.6%. LDB does not perform as well as the heuristic PRIO2.

The third heuristic methodology tested is the GA method described in Section 4.2. The following parameter values were used (see [21] for definitions): a population size of 312, a maximum of 150 generations, 15 clones, 36 immigrants, 54 mutation chromosomes, a mutation rate of 0.5 and range of 0.5, crossover probability of 0.7, and tournament selection with $t = 2$. Each problem was solved 10 times using a different initial random number seed. The first implementation, RKGA1, uses the RKGA described in Section 4.2. Table 3 shows the minimum, average, and maximum over the 10 seeds. Across all of the problems the average deviation from the lower bound was 2.3% and from the best known solution was 1.2%. The average computation times for the 50, 75, and 100 operation problems are 37, 65, and 100 seconds respectively on a Sun Sparc 20.

A second GA implementation, RKGA2, combines PRIO2 with the RKGA. This method "seeded" the RKGA with the best solution found by PRIO2. In this context, seeded means that one chromosome in the initial population received random key values that corresponded to the operation finish times for the best solution found by PRIO2. This seeded chromosome affected the GA search because it could be selected for crossover in subsequent generations.

The results for this method are shown in Table 3. The results are 0.6% better, on average, than RKGA1. This method required less computation time on average, 36, 64, and 88 seconds respectively on the three problem instances, because it solves some of the instances optimally in only a few seconds.

For P(2) the proposed heuristic methods found good solutions. The priority rule heuristic found solutions that were on average less than 3% above the lower bound and 2% above the best known solution. The combination of the priority rule heuristic and the RKGA found reduced these values to less than 2% and 1% respectively. Overall, the combination of the priority rule heuristics and GA methods found good solutions in a reasonable amount of computation time.

6 Conclusions and Further Research

In this paper we introduce some of the unique scheduling problems that arise for Parallel Machine Tools. The capability of these machines to perform simultaneous operations on single and multiple parts creates constraints that are not found in other problems discussed in the literature. Two types of solution methodologies are introduced for these problems. The first utilizes priority dispatching rules and utilizes a delay factor concept in order to enhance the quality of the developed solutions. The second utilizes a genetic algorithm based on the random keys encoding. Computational tests are presented for several example problems. The priority rule based heuristic finds good solutions to the example problems. The average deviation from the lower bound is less than 3% and bests [17], the only algorithm 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 as the solutions found are within 2% of the lower bound on average for P(2). However, this method does require more computation time than the priority rule heuristic method. The computation times are small enough that it is still possible to use this method for real-time control.

In future research, we will explore several problem extensions to P(2). The first extension relaxes assumption IV of Section 3— operations have already been assigned to PMLs. This creates another level of decisions in the model but should only require minor changes in the genetic algorithm model. The second is relaxing assumption III of Section 3 — 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. Relaxing this assumption significantly increases the complexity of the model but also makes the model more realistic. The third extension will be to include the assignment of tooling to turrets in the model.

References

[1] Baker, K. [1974], **Introduction to Sequencing and Scheduling**, Wiley.

[2] Bean, J. C. [1994], "Genetics and Random Keys for Sequencing and Optimization," **ORSA Journal on Computing**, Vol. 6, No.2, 154-160.

[3] Bell, C. E. and K. Park [1990], "Solving Resource-Constrained Project Scheduling Problem by A* Search," **Naval Research Logistics**, Vol. 37, 61-84.

[4] Bell, C. E. and H. Han [1991], "A New Heuristic Method in Resource-Constrained Project Scheduling," **Naval Research Logistics**, Vol. 38, 315-331.

[5] Boctor, F. F. [1990], "Some efficient multi-heuristic procedures for resource-constrained project scheduling," **European Journal of Operational Research**, Vol. 49, 3-13.

[6] Christofides, N., R. Alvarez-Valdes, and J. M. Tamarit [1987], "Project scheduling with resource constraints: A branch and bound approach," **European Journal of Operational Research**, Vol. 29, 262-273.

[7] Davis, E. W. and J. H. Patterson [1975], "A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling," **Management Science**, Vol. 21, No. 8, 944-955.

[8] Davis, L. [1991], **Handbook of Genetic Algorithms**, (ed. L.), Van Nostrand.

[9] Demeulemeester, E. and W. Herroelen [1992], "Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem." **Management Science**, Vol. 38, No. 12, 1803-1818.

[10] Elsayed, E. A. and N. Z. Nasr [1986], "Heuristics for resource-constrained scheduling," **International Journal of Production Research**, Vol. 24, No. 2, 299-310.

[11] Garey, M. R., D. S. Johnson, and R. Sethi [1976], "The complexity of flowshop and jobshop scheduling," **Mathematics of Operations Research**, Vol. 1, 117-129.

[12] Giffler, B. and Thompson, G. L. [1960], "Algorithms for solving production scheduling problems." **Operations Research**, Vol. 8, 487-503.

[13] Goldberg, D. E. [1989], **Genetic Algorithms in Search Optimization and Machine Learning**, Addison Wesley.

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

[15] Khattab, M. M. and F. Choobineh [1991], "A new approach for project scheduling with a limited resource," **International Journal of Production Research**, Vol. 29, No. 1, 185-198.

[16] Levin, J. and D. Dutta [1992], "Computer-Aided Process Planning For Parallel Machines," **Journal of Manufacturing Systems,** Vol. 11, 79-92.

[17] Levin, J., D. Dutta, J. C. Bean [1993], "PMPS: A Prototype CAPP System for Parallel Machining," **ASME Journal of Engineering for Industry**, to appear, May, 1996.

[18] Michalewicz, Z. [1994], **Genetic Algorithms + Data Structures = Evolution Programs**, 2nd Ed., Springer-Verlag.

[19] Morton, T. E. and D. W. Pentico[1993], **Heuristic Scheduling Systems**, John Wiley and Sons.

[20] Norman, Bryan A. [1995], "Scheduling Using the Random Keys Genetic Algorithm," Unpublished PhD. Dissertation, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, 48109-2117.

[21] Norman, Bryan A. and J. C. Bean, [1994], "Random Keys Genetic Algorithm for Job Shop Scheduling," Technical Report 94-5, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, 48109-2117.

[22] Olaguíbel, R. A. and J. M. T. Goerlich [1989], "Heuristic Algorithms For Resource-Constrained Project Scheduling: A Review And An Empirical Analysis," **Advances in Project Scheduling**, (ed. R. Slowiński and J. Weglarz), Elsevier, 113-134.

[23] Pinedo, M. [1995], **Scheduling Theory, Algorithms, and Systems**, Prentice Hall.

[24] Storer, R. H., S. D. Wu, and R. Vaccari [1992], "New Search Spaces for Sequencing Problems With Application to Job Shop Scheduling," **Management Science**, Vol. 38, No. 10, 1495-1509.

[25] Storer, R. H., S. D. Wu, and I. Park [1992], "Genetic Algorithms in Problem Space for Sequencing Problems," **Proceedings of a Joint US-German Conference on Operations Research in Production Planning and Control**, 584-597.

[26] Talbot, F. B. and J. H. Patterson [1978], "An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems," **Management Science**, Vol. 24, No. 11, 1163-1174.

[27] Ulusoy, G. and L. Özdamar [1994], "A constraint-based perspective in resource constrained project scheduling," **International Journal of Production Research**, Vol. 32, No. 3, 693-705.

[28] Ulusoy, G. and L. Özdamar [1989], "Heuristic Performance and Network/Resource Characteristics in Resource-constrained Project Scheduling," **Journal of the Operational Research Society**, Vol. 40, No. 12, 1145-1152.

[29] Yip-Hoi, D. [1994], Personal communication.

[30] Yip-Hoi, D. and D. Dutta [1993], "Issues in Computer-Aided Process Planning for Parallel Machines," **Advances in Design Automation**, American Society of Mechanical Engineers, Design Engineering Division (Publication) DE v 65 pt 1 1993. ASME,153-161.