International Journal of Production Research

ISSN: 0020-7543 (Print) 1366-588X (Online) Journal homepage: http://www.tandfonline.com/loi/tprs20

Scheduling preventive maintenance on a single CNC machine

Sinan Gurel & M. Selim Akturk

To cite this article: Sinan Gurel & M. Selim Akturk (2008) Scheduling preventive maintenance on a single CNC machine, International Journal of Production Research, 46:24, 6797-6821, DOI: 10.1080/00207540701487833

To link to this article: http://dx.doi.org/10.1080/00207540701487833

Published online: 31 Oct 2008.

International Journal of Production Research,
Vol. 46, No. 24, 15 December 2008, 6797–6821

Taylor & Francis
Taylor & Francis Group

Scheduling preventive maintenance on a single CNC machine

SINAN GUREL and M. SELIM AKTURK*

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

(*Revision received March 2007*)

In this study we attempt to deal with process planning, scheduling and preventive maintenance (PM) decisions, simultaneously. The objective is to minimize the total completion time of a set of jobs on a CNC machine. During the process planning, we decide on the processing times of the jobs which are controllable (i.e. they can be easily changed) on CNC machines. Using shorter processing times (higher production rates) would result in greater deterioration of the machine, and we would need to plan more frequent PM visits to the machine, during which it would not be available. Therefore, the selected processing times determine not only the completion times but also the PM visit times. We first provide optimality properties for the joint problem. We propose a new heuristic search algorithm to determine simultaneously the processing times of the jobs, their sequence and the PM schedule.

Keywords: Scheduling; Preventive maintenance; CNC machines; Total completion time; Heuristic search

1. Introduction

This study is an attempt to develop an integrated decision support system for preventive maintenance (PM) management, machine scheduling and process planning. We consider a single CNC machine, on which we can control the processing times by setting the machining conditions such as cutting speed and feed rate. We basically aim to answer the questions of when to make a PM visit to a machine, in what sequence to complete the jobs and at what processing times to operate them, simultaneously. The objective to minimize is the total completion time. To the best of our knowledge, in the literature and in current industry practice, these issues are considered either independent of each other or one being just a constraint for the others. Therefore, we introduce a model to combine these decisions by considering their effects on each other.

In the scheduling literature, most of the studies considering PM visits usually assume predefined PM visit times which form availability constraints during which no jobs can be operated on the machines. In practice, we can usually decide when to make a PM visit to a machine. Usually, a PM visit schedule is a decision to make rather than a constraint to satisfy. In this paper, we consider PM visit times as

*Corresponding author. Email: akturk@bilkent.edu.tr

International Journal of Production Research
ISSN 0020–7543 print/ISSN 1366–588X online © 2008 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/00207540701487833

decisions to be made in conjunction with machine scheduling decisions. Also, most of the literature and industry practice consider PM as an activity independent of the selected operating conditions on the machine, and they typically assume that these activities are made periodically (e.g. the time between two consecutive PM visits is fixed). However, the selected operating conditions (or production rates) influence the maintenance requirements of the system. If you run a machine under harsh conditions, you need to overhaul it more frequently. We employ a condition-based maintenance model that schedules the PM visits on a CNC machine by considering the selected processing times of the jobs on the machine.

A recent study that integrates process planning and condition-based maintenance scheduling decisions is that of Koomsap *et al*. (2005). They propose a model that collects information on the current condition of a machine in order to estimate its remaining useful lifetime. Their model uses the remaining useful lifetime information to revise the operating parameter selection decisions and/or the maintenance scheduling decisions for the machine. In this paper, we present a model that integrates process planning decisions with PM visit decisions via a condition parameter for the machine. In our model, the selected processing times (operating conditions) for the jobs determine the condition parameter value for the machine which is used to schedule PM visits to the machine.

We propose a model that makes scheduling, processing time selection and PM visit decisions simultaneously. Our model involves the effects of each decision on the others in order to minimize a scheduling objective. Such a decision support system can also be used as a tool in preventive maintenance management applications of computerized maintenance management systems (CMMS). MAXIMO is a CMMS where we can use our model for the purpose of integrating the work order generation process for the PM visits with the machine scheduling and process planning decisions.

In the scheduling literature, machine availability has received considerable attention. Adiri *et al*. (1989) study minimizing the flow time on a single machine with a single breakdown. They show that, for the case of a concave probability distribution for a breakdown, the shortest processing time first (SPT) rule minimizes the flow time. The SPT rule orders the jobs in ascending order of their processing times. They also show that the SPT rule minimizes the flow time under multiple breakdowns with an exponential distribution. For the deterministic case in which the breakdown times are known, they prove that finding a schedule that has a flow time less than a given value is NP-complete. Lee and Liman (1992) also study the deterministic case and find a tight worst-case bound of 2/7 for the SPT rule. For a recent review of studies on machine scheduling with availability constraints, we refer to Lee (2004).

There are also studies (similar to this study) that consider the start time of a non-availability period as a decision variable. Lee and Leon (2001) consider the decisions of when to schedule the rate modifying activity and sequence of jobs to optimize different regular scheduling measures, including total completion time, when there is a single maintenance period. Graves and Lee (1999) study the same problem and show that if there is exactly one maintenance period, then the total completion time problem can be solved polynomially by a modification of the SPT rule. Therefore, the main decision is to determine the set of jobs before and after the maintenance activity. Qi *et al*. (1999) study the total completion time of jobs

on a single machine with multiple maintenance intervals of variable duration, and prove that this problem is NP-hard in a strong sense. Akturk *et al*. (2004) and Qi (2007) establish the theoretical worst case performance of the SPT rule for this particular problem. Cassady and Kutanoglu (2003, 2005) also consider the preventive maintenance planning and production scheduling decisions simultaneously to minimize the total weighted tardiness and total expected weighted completion time, respectively. Jeong *et al*. (2006) add a diagnosis module in addition to maintenance planning and scheduling modules, and present an integrated decision support system for electronics manufacturing systems.

In contrast to these studies we deal with a scheduling problem where we can control the processing times of the jobs and the PM visit times (unavailability periods) on the machine simultaneously. It is important to note that the time between two consecutive PM visits should be shorter if we prefer higher production rates (shorter processing times), since this applies more cutting power and temperature on the machine. Controllable processing times have been receiving increasing attention in the recent scheduling literature. Hoogeveen (2005) gives a review on controllable processing times in multi-objective scheduling problems.

In the next section we give the problem definition and discuss the integrated approach. In section 3 we present the optimality properties for the problem. In section 4 we give the proposed heuristic search algorithm and apply it to a numerical example. In section 5 we discuss the computational results. Finally, we give the concluding remarks in section 6.

2. Problem definition

We have N jobs to be processed on a single CNC machine. We can control the processing times by setting the cutting speed and feed rate. However, selecting the cutting speed and feed rate is subject to technological constraints such as maximum applicable machining power, maximum allowable surface roughness and available tool life. Kayan and Akturk (2005) discuss the machining conditions selection problem for CNC turning machines and show that these technological constraints imply a lower bound on the processing time of a turning operation. The overall problem is to decide the processing time of each job, and to schedule the set of jobs and the required PM visits based on the selected machining conditions (i.e. cutting speed and feed rate), simultaneously. The objective is to minimize the total completion time of the jobs. Note that since the machine is not available during the PM visits, how many PM visits to schedule and when to schedule them are critical decisions for the total completion time objective. The notation for the rest of the discussion is as follows.

p_{i}	processing time of job i (min)
p_i^{l}	processing time lower bound of job i (min)
$M(p_i)$	PM priority index for job i at processing time p_i
C_{pm}	cost of a preventive maintenance visit (\$/visit)
T_{PM}	duration of a PM visit to the machine (min)
A,B,T,k,r	PM index function related parameters

X_{ij} 1 if job i is scheduled at position j, and 0 otherwise
Y_{kl} 1 if there is at least one PM visit between positions k and l

In this study, we introduce a new PM policy that uses processing time data while making PM visit scheduling decisions. This PM approach considers the fact that, as the production rate of a CNC machine increases (processing times decrease), we have to plan more frequent PM visits. This implies incurring more PM costs. In the existing PM approaches, one of the important parameters is the fixed and known mean time to failure (MTTF), which comes from a certain probability distribution function. This makes sense if we always run the CNC machine with the same cutting speed and feed rate for every manufacturing operation regardless of the work material, surface finish requirements, etc. But, in reality, we can change the processing times to optimize some performance measure. As a result, for a given operating time period T (which could be one week or one month depending on the available past shop floor data to estimate the related parameters), we can use the following PM cost function given by Akturk and Gurel (2007):

$$\text{PM cost} = A + B \cdot r^k \tag{1}$$

In the PM cost function, r is the production rate of the machine and A is the PM cost of the machine for time period T if no job is processed on the machine. This cost is due to the fact that we need to check and maintain the machine even when it is idle in order to keep it ready to operate when needed. When $r=0$ we pay a cost of A for maintenance. A, B and k are dependent on T and on the cost of a single PM visit, C_{PM}, which is assumed to be constant. An older machine would have higher B and k since it would require more frequent PM visits. We can determine the constant parameter B and exponent k empirically based on past shop floor data. The PM cost function is increasing and convex, so we have $B>0$ and $k \geq 1$. Assuming that the processing times of the jobs operated during T are all p, we can express the production rate of the machine as $r=1/p$. Then, the PM cost function can be written in terms of the processing time as follows:

$$\text{PM cost} = A + B/p^k. \tag{2}$$

Since we have the total PM cost for the operating period of length T, we can find the PM cost per job by dividing the total PM cost by the number of jobs completed in that period, which is T/p,

$$\text{PM cost per job} = \frac{A \cdot p^k + B}{T \cdot p^{(k-1)}}. \tag{3}$$

However, in practice, each job could have a different processing time, therefore we have to find the contribution of each job to the PM requirement of the machine, which is given by the ratio of its PM cost to the cost of a PM visit, C_{PM}. This is a value between 0 and 1 and gives a measure of the portion of a PM visit cost due to the considered job. We will call this measure the PM priority index of a job and the general expression for the PM priority index of job i is

$$M(p_i) = \frac{A \cdot p_i^k + B}{C_{\text{PM}} \cdot T \cdot p_i^{(k-1)}}. \tag{4}$$

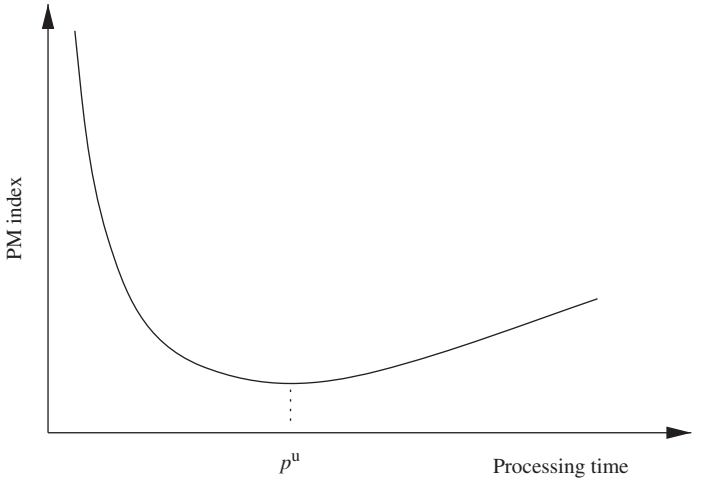

Figure 1. PM index versus processing time.

The shape of the PM index function by processing time is given in figure 1.

This index is calculated for each job that will be processed on the CNC machine and it indicates the PM requirement of the machine caused by this particular job. The sum of the PM indices of the jobs completed since the last PM visit is an indicator of the current PM need of the machine. When it reaches 1, then we need to schedule a PM visit to the machine, i.e. the sum of the PM indices of the jobs between two consecutive PM visits cannot exceed 1. Consequently, the proposed PM index can be used to provide an intelligent CNC machine degradation assessment, since it provides a condition parameter to be monitored to schedule PM visits to the CNC machine.

The problem is to find the optimal job and PM visit schedule and processing times of the jobs so as to minimize the total completion time of the schedule. The mathematical model for the problem is

$$\min \sum_{i=1}^{N}\sum_{j=1}^{N}(N-j+1)X_{ij}\cdot p_i \tag{5}$$

$$+\, T_{\mathrm{PM}}\sum_{j=1}^{N-1}(N-j)Y_{j(j+1)}, \tag{6}$$

subject to

$$\sum_{j=1}^{N} X_{ij} = 1 \quad i = 1,\ldots,N \tag{7}$$

$$\sum_{i=1}^{N} X_{ij} = 1 \quad j = 1,\ldots,N, \tag{8}$$

$$(1 - Y_{kl})\sum_{j=k}^{l}\sum_{i=1}^{N} X_{ij}\cdot M(p_i) \le 1 \quad k = 1,\ldots,N-1,\ l = k+1,\ldots,N, \tag{9}$$

$$Y_{kk+1} \le Y_{jk+1},\ k = 1,\ldots,N-1, \quad j = 1,\ldots,k-1 \tag{10}$$

$$p_i \geq p_i^{\mathrm{l}}, \ i = 1, \ldots, N, \tag{11}$$

$$X_{ij} \in \{0, 1\} \quad \text{and} \quad Y_{kl} \in \{0, 1\}, \quad i, j, k = 1, \ldots, N, \ l = k + 1, \ldots, N. \tag{12}$$

The objective function of the problem is composed of two parts. The first part (5) calculates the total processing time effect. The processing time effect of a job is its processing time plus its contribution to the completion times of all succeeding jobs. If we increase the processing time of a job by one unit, then the completion time of the job itself and the completion times of all succeeding jobs increase by one unit. The second part (6) calculates the total PM effect. The PM effect of a PM visit is the contribution of its duration to the completion times of all succeeding jobs. If we insert a PM visit between two jobs in the schedule, then all jobs succeeding the inserted PM visit are shifted backward and their completion times are increased by the duration of the PM visit. Between the two terms in the objective function there is an interesting trade-off that we have to deal with while solving the problem. If we decrease the processing times of the jobs to improve the processing time effect, then we need to have more PM visits, which increases the PM effect. Therefore, we cannot minimize both terms at the same time. This observation motivates the heuristic search algorithm which will be described in section 4. Constraint sets (7) and (8) are the assignment constraints that guarantee that each job is assigned to a position and each position is assigned to a single job. Constraint sets (9) and (10) ensure that the sum of PM indices between two PM visits does not exceed 1. Constraint set (11) sets a lower bound on the processing time of each job due to the machine and job related technological constraints.

What makes this total completion time problem interesting and difficult is the PM constraints (9) and (10). If we did not have these two sets of constraints, then we could easily set the processing time of each job to its processing time lower bound and order the jobs by the SPT rule. However, processing a job at different processing times has different consequences for the machine. In order to decrease the processing time, we need to run the machine at higher power due to the increased cutting speed and feed rate. This means greater force and higher temperature acting on the machine so that more wear takes place on the machine and the risk of sudden failure increases so that the machine requires more frequent PM visits.

The total completion time problem with fixed processing times and unavailability constraints such that a machine can operate at most $T > 0$ hours between two PM visits was shown to be NP-hard by Qi *et al.* (1999) and Akturk *et al.* (2004). In this study, we have controllable processing times and employ a nonlinear PM index function for PM scheduling decisions, therefore we deal with a more difficult problem. However, we have some information concerning the characteristics of the optimal solution, as discussed in the next section.

3. Optimality properties

For the total completion time problem with availability constraints and index-based PM scheduling decisions, we now present some optimality properties. The first is related to the optimal job sequence in a PM block (the set of jobs between two consecutive PM visits).

Property 3.1 (SPT in a PM block)**:** Between two consecutive PM visits, the SPT rule is optimal.

We can prove this property by pairwise interchanging adjacent jobs to see that a sequence not satisfying this property cannot be optimal. This property is common for the total completion time problems in Qi *et al*. (1999) and Akturk *et al*. (2004).

Property 3.2 (PM block utilization)**:** In an optimal schedule, for each PM block J, either all jobs in the block must have $p_i = p_i^{\mathrm{l}}$ or the sum of PM indices of the jobs in block J must be 1, i.e. $\Sigma_{i\in J} M(p_i) = 1$.

Proof: Assume that the opposite holds, i.e. $p_i > p_i^{\mathrm{l}}$ for some job $i \in J$ and $\Sigma_{i\in J} M(p_i) < 1$. Then, the objective can be improved by decreasing p_i without violating the total PM index constraint for the block, which proves the property. □

Property 3.2 states that, in an optimal schedule, for each PM block, either the processing time of each job in the block equals its lower bound, which means processing times cannot be decreased further, or the total PM index of the block equals 1, which means the block is fully utilized. We can illustrate this property by a numerical example. Consider a job i with $p_i = 1$ and $M(p_i) = 0.2$ in a PM block with a total PM index of 0.95. Suppose that when $p_i = 0.9$, then $M(p_i) = 0.25$. If we set $p_i = 0.9$, then the total PM index of the block is 1 and the total completion time of the schedule is decreased.

Property 3.3 (average job time)**:** The PM blocks in an optimal schedule should be in ascending order of their average processing times. Suppose that D_i and D_j are the total processing times, and n_i and n_j are the number of jobs in blocks i and j, respectively. In an optimal schedule, block i precedes block j, if $(D_i + T_{\mathrm{PM}})/n_i \leq (D_j + T_{\mathrm{PM}})/n_j$.

We can again prove this property by a pairwise interchange argument of two blocks. This property is also similar for the problems in Qi *et al*. (1999) and Akturk *et al*. (2004).

Property 3.4 (limits for the processing time of a job)**:** In an optimal solution, each job must satisfy the inequality $p_i^{\mathrm{l}} \leq p_i \leq \max(p_i^{\mathrm{l}}, p^{\mathrm{u}})$ where p^{u} is the processing time that gives the minimum PM index.

Proof: The left inequality above is a feasibility condition for the problem. The right inequality is an optimality condition. As p_i is increased beyond p^{u}, $M(p_i)$ increases. Increasing both p_i and $M(p_i)$ increases the total completion time. Therefore, we should not allow $p_i > p^{\mathrm{u}}$ unless $p_i^{\mathrm{l}} > p^{\mathrm{u}}$. □

Qi *et al*. (1999) and Akturk *et al*. (2004) proved some optimality properties for the total completion time problem with a maintenance or tool change constraint on a single machine. These two studies assume that maintenance or tool change should take place on a machine without exceeding a predefined operation time T. Our problem is different than these two problems in the sense that they assume that a job contributes to the PM need or tool need of a machine in proportion to its processing time, whereas we assume, in contrast to the current studies, that smaller processing time values (or higher cutting speeds and/or feed rates) may lead to a higher maintenance requirement since more force will be exerted on the CNC

machine. Thus, they have an optimality property that does not apply to our problem. This property is as follows. In an optimal solution, $n_i \geq n_j$ for any two PM blocks i and j such that i precedes j. This property is not applicable to our problem, since the PM index of a job in our problem may be higher for a job with a shorter processing time. A counterexample for our problem can be generated which violates this property. Suppose that we have five jobs assigned to two blocks. Block 1 has two jobs with $p_i = 1$ and $M(p_i) = 0.5$ and block 2 has three jobs with $p_i = 2$ and $M(p_i) = 1/3$. For the given schedule, $n_1 < n_2$ and the total completion for these two blocks is 30. If we interchange the two blocks, we obtain a total completion of 33 with the first block in the sequence having more jobs than the second. This demonstrates that our problem is quite different than those studied in the literature.

4. Heuristic search algorithm

Considering the optimality properties given in the previous section and the trade-off between the processing time effect and the PM effect components of the objective function, we propose a heuristic search algorithm for the problem. Our search algorithm includes two main parts: the branching process and the improvement process. The branching process enumerates several points in the solution space in a systematic way in order to find the best solution.

4.1 *Branching process*

The branching process starts with an initial schedule and applies a search procedure to achieve an improved schedule. We form the initial schedule by setting $p_i = p_i^{\downarrow}$ for each job i and ordering the jobs by the SPT rule. We add the PM visits into the schedule by using the index-based PM policy as discussed in section 2. This is the schedule in which the processing time effect is minimum. However, the PM effect is too great due to there being too many PM visits. Hence, the branching process aims to find alternative schedules with smaller PM effects. To do this, the branching process moves jobs forward to the preceding PM blocks in order to reduce the number of PM visits. Moving a set of jobs forward to a PM block requires increasing the processing times of the jobs inside the block due to the PM index constraints. Therefore, improving the PM effect results in an increased processing time effect. The branching process generates alternative schedules by moving different sets of jobs to different PM blocks iteratively and aims to achieve a solution where the processing time effect and the PM effect components are balanced to minimize the total completion time.

Having formed the initial schedule, the branching process first moves the jobs scheduled immediately after the first PM visit to the first PM block. By moving different sets of jobs forward to the first PM block each time, the branching process generates C new schedules. We call these new schedules child schedules of the parent (initial) schedule. The branching process forms the first child schedule of the initial schedule by moving a set of jobs with total PM index of at most α to the first PM block. Similarly, the kth child schedule is formed by moving a set of jobs with total PM index of at most $k\alpha$. If the total PM index of the jobs that can be moved is less

than $k\alpha$, then we move all of them regardless of their total PM index. We call the generation of child schedules by moving jobs to the first PM block Stage 1.

When a set of jobs is moved to the first PM block, the total PM index of the block exceeds 1, since the block is already fully utilized due to property 3.2. Then, for the first PM block of each child schedule we must find the optimal job sequence and corresponding processing times that minimize the total completion time of the new schedule while the total PM index of the first block is less than or equal to 1. The main aim of this subproblem is to minimize the total processing time effect of the jobs in the PM block. In section 4.2 we propose three alternative algorithms to solve this subproblem.

Provided that we have an algorithm to solve the PM block subproblem, the branching process solves the subproblem for the first PM block of each new child schedule formed. At the end of Stage 1, the branching process selects P child schedules with minimum total completion times as the new parent schedules. We use these parent schedules to generate new child schedules in Stage 2. In Stage 2, we generate child schedules by moving the jobs scheduled after the second PM visit forward to the second PM block. From each parent schedule, we again form C new child schedules. This time we solve the subproblems for the second PM blocks of the child schedules. Then, we select P parent schedules from these child schedules. In Stage 3, we generate child schedules by moving jobs immediately scheduled after the third PM visit forward to the third PM blocks of the parent schedules. Each time the branching process moves jobs forward to a PM block, the PM effect component improves but the processing time effect becomes worse. The branching process continues in this way until no parent schedules can be found. For instance, if all the child schedules in Stage 2 have at most two PM blocks, then no child schedule can be a parent for Stage 3. The branching process enumerates several alternative schedules in this way and outputs the best schedule it achieves. The branching process that we have described above with parent solutions and child solutions, roughly resembles a beam search method, but, differently, in our method each node represents a complete schedule. An example of the application of beam search to scheduling problems can be found in Ow and Morton (1988).

To illustrate child formation in the branching process, we give the following numerical example. We have eight jobs to be scheduled, and the PM index function is defined as $M(p_i) = (5p_i^{2.5} + 1800)/(3750p_i^{1.5})$. PM visit duration is $T_{\mathrm{PM}} = 4$. The number of child schedules to be generated from each parent is $C = 3$ and the number of parents to be selected at each stage is $P = 2$. The set of jobs to be moved at each step is determined by $\alpha = 0.4$. The initial schedule for the example is given in figure 2, where the processing time of each job is set to its lower bound and the jobs are ordered by the SPT rule. In figure 2, processing times and corresponding

Initial schedule (total completion time = 82)

	1	2		3	4	5		6	7	8
p_i	1.0	1.1		1.2	1.4	1.6		1.6	1.6	1.7
M_i	0.48	0.42		0.37	0.29	0.24		0.24	0.24	0.22

Figure 2. Initial schedule for the branching process.

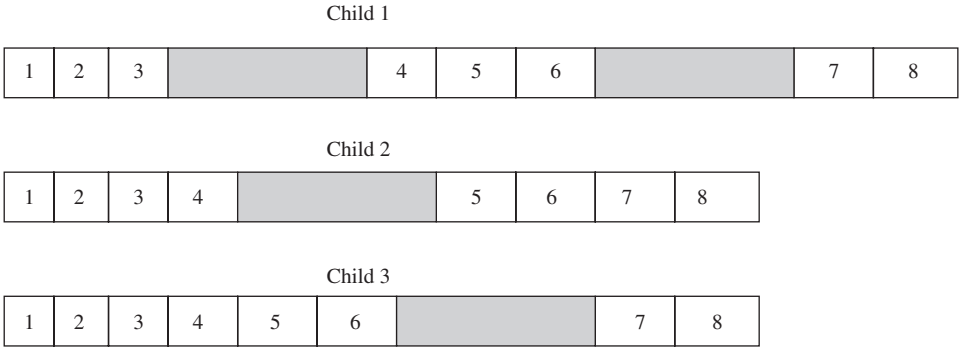

Figure 3. Child schedules in Stage 1 (before solving the subproblem).

PM indices are given below the Gantt chart. Each shaded time interval is a PM visit. For the initial schedule, the total completion time is 82, which is the sum of the processing time effect and the PM effect, which are 46 and 36, respectively.

In the numerical example, in Stage 1, we generate three child schedules from the initial schedule. To form the first child schedule, we move a set of jobs with a total PM index of at most 0.4, e.g. we move job 3 with a PM index of 0.37 forward to the first PM block. We form the second child schedule by moving a set of jobs with maximum total PM index of 0.8. Then, we move jobs 3 and 4 forward to the first PM block. Finally, to form child 3 we move jobs 3, 4, 5 and 6 with a total PM index of $1.12 < 1.2$. Figure 3 shows the allocation of jobs to the PM blocks in the child schedules achieved in Stage 1. Since the PM block subproblems are not yet solved, the total PM index of the first block in each child schedule exceeds 1 and the current schedules in figure 3 are infeasible. Figure 3 also shows how the remaining jobs succeeding the first block are scheduled. As can be seen, moving jobs forward to the first block reduces the number of PM visits required in child 2 and child 3. It is also obvious that, with each new generated child schedule, the PM effect decreases or remains the same.

In this subsection we have discussed the proposed branching process to search the solution space. In the next subsection we will discuss the PM block subproblem and present alternative solution approaches for the subproblem. We will also extend the numerical example to illustrate the applications of different solution approaches for the subproblem.

4.2 *PM block subproblem*

In the branching process, when we move a set of jobs forward to a PM block and form a child schedule, we need to change the processing times of the jobs in the PM block so that the total PM index of the jobs in the block is less than or equal to 1. We need to find the optimal schedule and the corresponding optimal processing times at the same time to minimize the total processing time effect of the jobs in a given PM block. We propose three heuristics to solve this problem, namely the LP-based method (LPB), the first update shortest method (FUS) and the total completion index method (TCI).

4.2.1 LP-based method (LPB). We first consider the case where there is a given job sequence in a PM block. The problem is to minimize the total processing time effect of the given n jobs in the block. The first job scheduled in the block is the kth job of the entire schedule. Then, the second job is the $(k+1)$th job of the schedule, and so on. In the model below, the subscript j in p_j and p_j^{l} indicates the jth job in the sequence in the PM block. The mathematical model for the block subproblem is as follows:

$$\min \sum_{j=k}^{n+k-1} (N-j+1)p_j \tag{13}$$

subject to

$$p_j \geq p_j^{\mathrm{l}} \quad j = k, \ldots, (n+k-1), \tag{14}$$

$$\sum_{j=k}^{n+k-1} M(p_j) \leq 1. \tag{15}$$

The objective function (13) is the sum of the processing time effects of all jobs in the block. Constraint set (14) ensures that the processing time of each job is greater than its lower bound. Constraint (15) ensures that the total PM index of the jobs in the block is less than or equal to 1. The above formulation is a separable nonlinear programming model, since each nonlinear term in constraint (15) is the PM index function of a distinct job j and formed of a single variable p_j. Furthermore, each term is a convex function of p_j since $M(p_j)$ is a convex function. We can solve the problem approximately as discussed by Bazaraa *et al.* (1993) by applying the simplex method to the piecewise linear approximation of the model.

In order to achieve a piecewise linear approximation of the above model, we generate a set of breakpoints for the PM index function $M(p_j)$. Using these breakpoints, the processing time p_j in the previous model can be expressed as

$$p_j = \lambda_{j1} \cdot \mu_{j1} + \cdots + \lambda_{jB_j} \cdot \mu_{jB_j},$$

where μ_{jl} are the breakpoints and λ_{jl} are the coefficients for job j such that $0 \leq \lambda_{jl} \leq 1$ for all l and $\lambda_{j1} + \cdots + \lambda_{jBj} = 1$. B_j is the number of breakpoints for job j. Here, μ_{j1} is the minimum processing time p_j^{l} for job j and μ_{jBj} is the highest breakpoint, which equals p^{u}. Then, after applying piecewise linear approximation, the model is as follows:

$$\min \sum_{j=k}^{n+k-1} \sum_{l=1}^{l=B_j} (N-j+1)\lambda_{jl} \cdot \mu_{jl}, \tag{16}$$

subject to

$$\sum_{j=k}^{n+k-1} \sum_{l=1}^{l=B_j} \lambda_{jl} \cdot M(\mu_{jl}) \leq 1, \tag{17}$$

$$\sum_{l=1}^{l=B_j} \lambda_{jl} = 1, \quad j = k, \ldots, n+k-1, \tag{18}$$

$$\lambda_{jl} \geq 0, \quad j = k, \ldots, n+k-1 \quad \text{and} \quad l = 1, \ldots, B_j. \tag{19}$$

Constraint (17) is the PM index constraint for the block. Constraint sets (18) and (19) ensure that the model chooses processing times that are convex combinations of the breakpoints.

Based on the above findings, we designed an LP-based algorithm for the PM block subproblem. The LPB algorithm first orders the jobs in ascending order of their p_i^{l}. This is due to property 3.1. The jobs must be in the SPT order at optimality, therefore it is more effcient to put the job that can have the minimum processing time in the first position. We solve the piecewise linear approximation model for this job sequence. If there are too many jobs in the block, then the PM index constraint may not be satisfied and the problem turns out to be infeasible. After solving the LP, the resulting schedule may not satisfy the SPT rule due to the breakpoints. In such a case, we re-sequence the jobs by the SPT rule and stop. We can express the LPB method as follows.

- **Step 1:** Sequence the jobs in the block in ascending order of their p_i^{l} (i.e. apply the SPT rule for p_i^{l} of the jobs).
- **Step 2:** Solve the piecewise linear approximation model for the block.
- **Step 3:** If the LP is infeasible, then stop, the block is infeasible. Else, if the achieved processing times from solving the LP model violate the SPT rule, then reorder the jobs by the SPT rule.

We applied the LPB method to the first blocks of the child schedules achieved in Stage 1 of the numerical example. We used the breakpoints (1.1, 1.2, 1.3, ..., 12.2, 12.3) for the piecewise linear approximation of the PM index function. We give the resulting child schedules in figure 4. It can be seen from figure 4 that, after adding new jobs to the first PM block, we need to increase the processing times of the jobs in the block. For the first child schedule, we have a total completion time of 77.9, which is better than the completion time of the initial schedule (82). For the second child, it is 72.3. Note that removing a PM visit improves the total completion time. For the

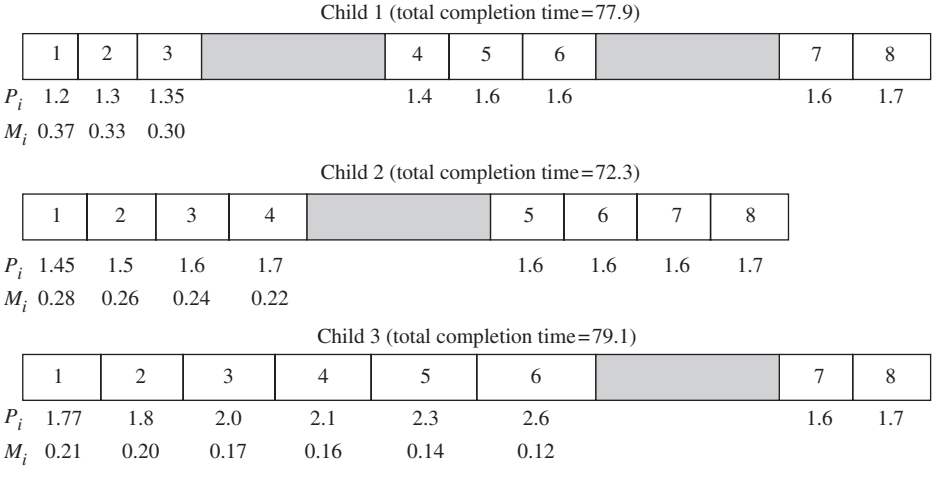

Figure 4. Child schedules in Stage 1 (LPB method applied).

third schedule it is 79.1, which is worse than the second child schedule. We have six jobs in the first block in child 3 and in order to satisfy the PM index constraint we need to increase the processing times of the jobs excessively and achieve a schedule that is worse than the second child schedule.

Since we need to solve the PM block subproblems many times in the branching process, calling the LP solver each time and solving the problem may be computationally inefficient. Therefore, we propose alternative approaches. The second approach is the first update shortest method (FUS), which is discussed in the next subsection.

4.2.2 First update shortest method (FUS). The FUS method first sets the processing time of each job to its lower bound (p_i^{l}) at which the PM index of the job is at a maximum. Then, in each iteration, it increases the processing time of the shortest job to the next breakpoint, until the total PM index of the block is less than or equal to 1. The breakpoints used by the FUS method are the same as the points used by the LPB method. The LPB method tries to minimize the processing time effect of the jobs, so that the position information for each job is included in the objective function. However, the FUS method ignores the position information of each job and just tries to keep the length of the PM block as short as possible. We give the steps of the FUS method below.

- **Step 1:** Set $p_i = p_i^{\mathrm{l}}$ for each job i, and order the jobs by the SPT rule.
- **Step 2:** While the total PM index of the block is greater than 1, carry out the following steps.
  - **Step 2.1:** Take the shortest job i and if $p_i < p^{\mathrm{u}}$, set $p_i = p_i'$, where p_i' is the next breakpoint for job i. Else, if $p_i \geq p^{\mathrm{u}}$, then the block is infeasible.
  - **Step 2.2:** Update $\sum_{j=k}^{n+k-1} M(p_j)$.
  - **Step 2.3:** If the resulting schedule violates the SPT rule, reorder the jobs by the SPT rule.

Since $M(p_j)$ is a convex function, the PM index decrease per unit processing time increase is greater for smaller processing times, so the FUS method always updates the shortest job in the block. It considers the makespan of the block rather than the processing time effect of each individual job. The FUS method is easier and faster to implement than the LPB method. We applied the FUS method for the third child of Stage 1 in the numerical example, and the resulting schedule is given in figure 5. When we compare the schedule achieved by the FUS method in figure 5 with child 3 in figure 4, we see that the LPB method achieved a better total completion time (79.1)

Child 3 (total completion time = 80.1)

	1	2	3	4	5	6		7	8
P_i	2.0	2.0	2.0	2.1	2.1	2.1		1.6	1.7
M_i	0.17	0.17	0.17	0.16	0.16	0.16			

Figure 5. Child 3 in Stage 1 (FUS method applied).

than the FUS method (80.1). In the next subsection we give the total completion index (TCI) method.

4.2.3 Total completion index method (TCI). The TCI method starts with the same initial solution as the FUS method and increases the processing time of a selected job at each iteration like the FUS method. The difference between the two methods is that the TCI method selects the job to be updated by considering the positions of the jobs. The TCI method calculates an index (C_i) for each job i which measures the change in the processing time effect per change in the PM index to be achieved by setting the processing time of job i to its next breakpoint. This index is denoted the total completion index and for job i at position k we can calculate the index as follows:

$$C_i = \frac{(N-k+1)(p_i' - p_i)}{(M(p_i) - M(p_i'))},$$

where p_i is the current processing time of job i and p_i' is the next breakpoint for job i. Obviously, the job with minimum C_i seems to be the best job to update. We give the steps of the TCI method below.

- **Step 1:** Set $p_i = p_i^{\mathrm{l}}$ for each job i, and order the jobs by the SPT rule.
- **Step 2:** While the total PM index of the block is greater than 1 do the following steps.
  - **Step 2.1:** Find job i (for which $p_i \leq p^{\mathrm{u}}$) with the minimum C_i. If no such job exists, then the block is infeasible. Else, set $p_i = p_i'$.
  - **Step 2.2:** If the resulting schedule violates the SPT rule, reorder the jobs by the SPT rule.
  - **Step 2.3:** Update the C_i indices of job i and all rescheduled jobs in Step 2.2.

The TCI method selects the job that gives the minimum processing time effect change per PM index change when we set its processing time to the next breakpoint. At each iteration, it reorders the jobs by the SPT rule and it recalculates the total completion indices of the jobs after reordering them. In figure 6 we give the child schedule 3 at Stage 1 achieved by applying the TCI method. Since this method uses the position information for the jobs, it gives a better total completion time (79.7) than the FUS method (80.1).

We have proposed three heuristics to solve the PM block subproblem. In the next subsection we will discuss the fine tuning approach. Fine tuning is an intensification method for the branching process and it generates alternative child schedules inside the branching process.

Child 3 (total completion time=79.7)

	1	2	3	4	5	6		7	8
P_i	1.8	1.9	2.0	2.2	2.3	2.3		1.6	1.7
M_i	0.20	0.19	0.17	0.15	0.14	0.14			

Figure 6. Child 3 in Stage 1 (TCI method applied).

4.3 *Fine tuning in the branching process*

As explained in section 4.1, in the branching process we generate child schedules by shifting jobs to an earlier PM block. The number of jobs to be shifted to form the kth child schedule of a parent schedule is determined by the PM index value $k\alpha$. In fine tuning, for a parent schedule achieved in the branching process, we select the child schedule with minimum total completion time, say child k. We generate two additional child schedules using the neighbour PM index values of $k\alpha$, i.e. $(k+1/2)\alpha$ and $(k-1/2)\alpha$. When $\alpha = 0.4$, and if we obtain the best child of a parent schedule with a PM index value of 0.8, then we also generate child schedules by moving sets of jobs with total PM index less than or equal to 0.6 and 1.0. Child schedule formation is the same as in the branching process such that if the total PM index of the jobs that can be moved is less than the requested level, then we move all the jobs to the considered PM block. Fine tuning looks for better alternatives in the neighbourhood of the best schedule achieved from a parent schedule. The motivation behind the fine tuning is the behaviour of the total completion times of the child schedules generated from a parent schedule. When we analyse the total completion time values of the child schedules of a parent schedule, we observe that, as k increases in $k\alpha$, the resulting total completion times of the child schedules first decrease and then increase. This is also the case for the child schedules in figure 4. This behaviour shows that looking for possible child schedules in the neighbourhood of the best child schedule may provide better schedules. Therefore, we apply fine tuning for each parent schedule at each stage of the branching process.

In the numerical example for Stage 1, by fine tuning we generated two more alternative child schedules. Since the best child schedule generated is child 2, we moved sets of jobs with total PM indices of 0.6 and 1.0 to form child 4 and child 5, shown in figure 7. For the first PM block of each new child schedule, we solved the PM block subproblem using the LPB method. Child 4 is the same as child 1, but child 5 is a new schedule with five jobs in its first PM block. We generated schedules with three, four and six jobs in the first block previously, but fine tuning generated the schedule with five jobs in the first block. After we applied the fine tuning step, we completed the child formation part of Stage 1 in the numerical example. The best schedule achieved is child 2 with a total completion time of 72.3.

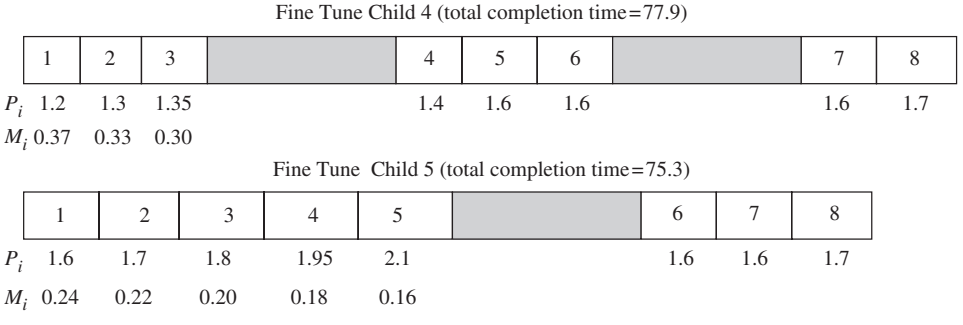

Figure 7. Child schedules generated by fine tuning.

Child 1 (total completion time=74)

	1	2	3		4	5	6	7		8
P_i	1.2	1.3	1.35		1.4	1.6	1.6	1.65		1.7
M_i	0.37	0.33	0.30		0.29	0.24	0.24	0.23		0.22

Child 2 (total completion time=72.6)

	1	2	3		4	5	6	7	8
P_i	1.2	1.3	1.35		1.42	1.6	1.8	2.1	2.7
M_i	0.37	0.33	0.30		0.29	0.24	0.20	0.16	0.11

Figure 8. Child schedules in Stage 2.

Next, in the numerical example, we select two parent schedules from the five child schedules. Child 2 is the one with the minimum total completion time (72.3), but it only has two blocks and therefore it cannot be a parent schedule for Stage 2. Similarly, child 3 and child 5 cannot be parents either. Since the schedules in child 1 and child 4 are identical, we have one parent schedule for the next stage. In Stage 2 we generated child schedules by moving jobs forward to the second block of the parent schedule. We generated two child schedules in Stage 2, which are given in figure 8. Since these schedules have at most three blocks, they cannot be selected as parents for Stage 3, and the branching process ends. The best schedule achieved by the branching process is child 2 in Stage 1 given in figure 4. Up to this point, we have discussed the branching process, the PM block subproblem and the fine tuning approach. In the next subsection we discuss the improvement methods that can be applied after the branching process.

4.4 *Improvement methods*

Motivated by the optimality properties given in section 3, we propose two improvement steps to be applied to the schedule achieved by the branching process. The first is the Last Block Improvement procedure, which will be discussed in the next subsection. The second is Block Rearrangement.

4.4.1 Last block improvement (LBI). The branching process may terminate with a schedule whose last block has a total PM index strictly less than 1. From property 3.2 we know that, in such a case, $p_i = p_i^1$ for each job i in the last block. In preceding blocks there may exist jobs that have greater processing times but smaller processing time lower bounds than some of the jobs in the last block. If we can find such job pairs, one from the last block and the other from the preceding blocks, then we can pairwise interchange these two jobs. This allows us to schedule the jobs with smaller processing time lower bounds in the last block. We set the processing times of the jobs in the last block to their lower bounds and if the total PM index of the last block exceeds 1 we solve the subproblem for the block. In this way we can improve the total processing time effect of the last block and improve the

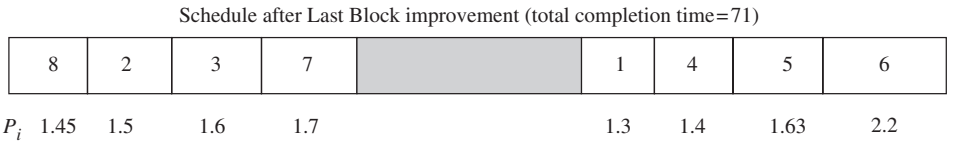

Figure 9. Schedule achieved by applying the LBI method.

total completion time of the schedule. The LBI method is motivated by property 3.2 (Block Utilization). If the last block of the schedule is fully utilized, the LBI method cannot achieve any improvement. Therefore, before applying the LBI method, we check the total PM index of the last block of the schedule and if it is less than $1-\gamma$, where γ is a user-defined value, we apply the method, otherwise we omit it.

We applied the LBI method to the best schedule (child 2 in Stage 1 of figure 4) achieved by the branching process. Jobs 1, 4 and 7, 8 are interchanged between blocks and the resulting schedule is given in figure 9. Observe that the processing times of 7 and 8 are set to 1.7 and 1.45 (the processing times for jobs 4 and 1 before interchanging), respectively. The processing times of 1 and 4 are set to 1 and 1.4, but then the total PM index of the last block exceeds 1, and we solved the block subproblem for the last block using the LPB method. After applying the LBI method, the total completion time is improved from 72.3 to 71. In the next subsection we describe the second improvement step, Block Rearrangement.

4.4.2 Block rearrangement. Block Rearrangement is the application of the block ordering rule given in property 3.3 of section 3. In this method, we first check if the PM blocks of the schedule are ordered in ascending order of their average processing times or not. If not, we order the blocks and then solve the PM block subproblems for the affected blocks, since re-sequencing the blocks changes the positions of the jobs in the schedule. We repeat the steps of reordering the blocks and solving the PM block subproblems until we achieve a schedule where the PM blocks are in ascending order of their average processing times. In the numerical example, for the schedule achieved by the LBI in figure 9, the average processing times for blocks 1 and 2 are 2.05 and 2.11, respectively. They satisfy the condition in property 3.3 and we do not need to re-sequence them.

Thus far, we have described the branching process, the fine tuning process and the improvement methods which form the main steps of our search algorithm. We have also discussed three alternative approaches to solve the PM block subproblem. We have discussed how each procedure can be applied on a numerical example. Using these methods given above we designed five different versions of the search algorithm. The first is the search algorithm using the LPB method. In the next subsection we describe the algorithms in logical sequence.

4.5 *Search algorithm using the LPB method (S_{LPB})*

The S_{LPB} algorithm uses the LPB method to solve the PM block subproblems that arise in the branching process, the last block improvement and the block rearrangement steps. The steps of the S_{LPB} algorithm are as follows.

- **Step 1** (finding the initial schedule)**:** Set $p_i = p_i^{\mathrm{I}}$ for each job i. Find a schedule by sequencing the jobs using the SPT rule and inserting the PM visits into the schedule where needed.
- **Step 2:** Set the algorithm parameter α.
- **Step 3** (branching process)**:** Set the initial schedule as a parent schedule for Stage 1 and as the best schedule achieved so far. Initialize the stage counter, $s = 1$.
  - **Step 3:1:** For each parent schedule in Stage s perform Steps 3.1.1 to 3.1.6.
    - **Step 3.1.1:** Generate C child schedules by moving forward the jobs of the total PM index $\alpha, 2\alpha, \ldots, C\alpha$ to the sth block of the parent schedule.
    - **Step 3.1.2:** Solve the PM block subproblem for the sth block of each child schedule using the LPB method.
    - **Step 3.1.3:** Select the child schedule with the minimum total completion time, denoted child k.
    - **Step 3.1.4** (fine tuning)**:** Generate two new child schedules from the parent schedule by shifting jobs of the total PM index $(k+1/2)\alpha$ and $(k-1/2)\alpha$ to the sth block.
    - **Step 3.1.5:** Solve the PM block subproblem for the sth block of each child schedule generated in Step 3.1.4.
    - **Step 3.1.6:** Update the best schedule achieved so far if a better schedule exists among the child schedules generated in Stage s.
  - **Step 3.2:** Select P parent schedules from the child schedules achieved in Stage s. If no child schedule qualifies to be a parent schedule for Stage $s+1$, then go to Step 4. Otherwise, go to Step 3.1.
- **Step 4** (last block improvement)**:** If the total PM index of the last block of the best schedule achieved so far is less than $1-\gamma$, then go to Step 4.1. Otherwise, go to Step 5.
  - **Step 4.1:** Starting from the first job of the last block, for each job k of the last block we search for a job i with the minimum p_i^{I} (in all PM blocks except the last block), such that $p_i^{\mathrm{I}} < p_k$. If found, interchange the positions of jobs i and k. Set $p_k = p_i$ and $p_i = p_i^{\mathrm{I}}$. Otherwise, go to Step 4.2.
  - **Step 4.2:** If the total PM index of the last block exceeds 1, solve the PM block subproblem for the last block using the LPB method.
- **Step 5** (block rearrangement): Evaluate the average processing time value for each PM block in the schedule.
  - **Step 5.1:** If the blocks are in ascending order of their average processing times, then go to Step 6. Otherwise, order the blocks in ascending order of average processing times and go to Step 5.2.
  - **Step 5.2:** Solve the PM block subproblems for all blocks using the LPB method and go to Step 5.
- **Step 6:** Report the best schedule achieved so far.

In Step 1 we form an initial schedule for the search algorithm. In Steps 3.1.1 to 3.1.3 we generate child schedules for a parent schedule in Stage s by moving jobs to its sth block and then solving the PM block subproblem using the LPB method. In Steps 3.1.4 and 3.1.5 we apply the fine tuning method using the best child schedule found in the previous steps (3.1.1 to 3.1.3). In Step 4 we apply the last block improvement procedure which utilizes the available PM capacity in the last block of the schedule. Finally, in Step 5, we apply the block rearrangement process iteratively, which sequences the PM blocks in ascending order of average processing times and solves the block subproblems using the LPB method.

The S_{LPB} algorithm employs the LPB method to solve the PM block subproblems. During the branching process, each time a PM block subproblem is to be solved, the S_{LPB} algorithm formulates an LP problem and calls an LP solver to solve it. Although the LPB method is more efficient in solving the subproblems, it may require too much computational effort, so we propose alternative versions of the search algorithm that use the FUS and TCI methods.

4.6 *Search algorithms using the FUS method*

In this subsection we give three different versions of our search algorithm. These algorithms mainly use the FUS method to solve the PM block subproblems, but two of them also utilize the LPB method. The first version is S_{FUSI}, which employs the FUS method to solve the PM block subproblems that arise in Steps 3.1.2, 3.1.5 and 4.2. In the block rearrangement procedure, S_{FUSI} omits Step 5.2 since the FUS method solves the block subproblems independent of the positions of the jobs. Therefore, the block rearrangement step (5.1) is applied only once.

Since the FUS method is not sensitive to the positions of the jobs but is computationally efficient to use, we propose another version (S_{FUSII}) of the search algorithm that applies the FUS method in the branching process, but utilizes the LPB method in the improvement steps. As for the S_{FUSI} algorithm, this algorithm employs the FUS method to solve the PM block subproblems in the branching process (Step 3.1.2) and the fine tuning process (Step 3.1.5). Differently, S_{FUSII} applies the LPB method to solve the block subproblem for each PM block of the best schedule achieved by the branching process in Step 3.2. The LPB method is more efficient than the FUS method in solving the block subproblems, but it requires more computational effort. Therefore, in the branching process we call the LPB method just for the final schedule achieved by the branching process, which can make a significant difference in the solution quality while requiring marginal additional computation time. The remaining steps of S_{FUSII} for the last block improvement and the block rearrangement are the same as for the S_{LPB} algorithm, therefore it utilizes the LPB method to solve the PM block subproblems that arise in these improvement procedures.

The last version of the FUS-based algorithms is S_{FUSIII}. This algorithm is the same as S_{FUSII} except that it includes a test to decide whether or not to apply the LPB method to solve the PM block subproblems in Steps 3.2 and 5.2. The test procedure roughly checks if an improvement is possible, if we use the LPB method on a PM block or not, in order to save computational time.

Test procedure for a PM block:

- **Step 1:** Starting from the first job in the block, find the first job i whose processing time can be set to its previous breakpoint. If found, then go to Step 2. Else, stop, LPB will not be applied.
- **Step 2:** Calculate the possible PM index loss and total completion time gain that would occur by setting job i to its previous breakpoint without changing its position.
- **Step 3:** Set j as the last position in the block.
- **Step 4:** Calculate the PM index gain and total completion time loss that would occur by setting the last job j in the block to its next breakpoint.
- **Step 5:** If the PM index loss for job i in Step 2 and the PM index gain for job j in Step 4 result in a feasible block with total PM index less than or equal to 1, and if the total completion time gain in Step 2 is greater than the total completion time loss in Step 4, stop; the LPB method will be applied to the block. Else, if j is the immediate successor of job i, then stop; no more jobs to be processed. Else, set $j=j-1$ and go to Step 4.

In Step 1 the test procedure finds a job in order to reduce its processing time to the previous breakpoint. The first available job is selected since the processing time decrease could result in a greater total completion time gain. In Step 2 we calculate the possible PM index loss and total completion time gain for the selected job. Then in Steps 3 to 5 the procedure looks for another job in order to increase its processing time to the next breakpoint so that the resulting total PM index of the block is still less than 1 but the total completion time objective is improved. This test is applied for all blocks of the schedule achieved in Step 3.2, starting from the first block up to a block for which it decides to apply LPB or ends with the last block. If the test decides to apply LPB to block i, then LPB is applied to block i and to all the blocks succeeding block i. The last version of the search algorithm, which uses the TCI method, is given in the next subsection.

4.7 *Search algorithm using the TCI method (S_{TCI})*

We have described four versions of the search algorithm which use the LPB and FUS methods. The last version (S_{TCI}) of the search algorithm employs the TCI method for PM block subproblems. The S_{TCI} algorithm is the same as S_{LPB} except that instead of the LPB method, it applies the TCI method to solve the PM block subproblems arising in Steps 3.1.2, 3.1.5, 4.2, and 5.2. In the next section we discuss the experimental design and give the computational results for these algorithms.

5. Computational results

In this section we present the computational results for the proposed search algorithms. We coded the algorithms in C language and compiled using the Gnu C compiler version 2.95.3. We ran the codes on a Solaris 2.7 operating system using a Sun HPC 4500 workstation with a 12×400 MHz UltraSPARC CPU and 3 GB memory. In coding the LPB method, we used the library routines of the CPLEX 9.1 commercial solver.

Table 1. Experimental design factors.

Factor	Definition	Level 1	Level 2
T_{PM}	PM duration	30	90
A	PM index function parameter	10	20
B	PM index function parameter	30	45

In the computational study we consider a CNC turning machine with a maximum applicable cutting power of 10 hp. Each job to be operated on this machine corresponds to a turning operation. The processing time lower bound (p_i^l) for a turning operation is determined by its cutting specifications, required cutting tool type and the surface quality. Therefore, for each replication we generated cutting specifications (diameter, length, depth of cut and required surface roughness) for the jobs randomly as discussed by Kayan and Akturk (2005). We considered problems with 2000 jobs, which corresponds to a 5-day production rate if the machine produces at a rate of one job/min on average. There are three experimental factors, given in table 1, which can affect the efficiency of the algorithms. The first experimental factor is the T_{PM} level, which is the PM visit duration for the machine. It influences the PM effect component of the total completion time objective. When $T_{PM} = 30$ we set $\alpha = 0.4$, else $\alpha = 0.6$. If the duration of a PM visit is long, then while forming the child schedules in the proposed search algorithm, moving more jobs forward could reduce the number of PM visits, therefore we prefer to use a higher α value. The other two experimental factors are A and B (the coefficients used in the PM index function), which affect the function behaviour. Other PM index function coefficients are $C_{PM} = 10$, $T = 2000$, and $k = 2.5$. We consider two levels for each factor, so we have a 2^3 full factorial design. We took 10 replications for each factorial setting. Another critical decision in the experimental study is the selection of breakpoints for the piecewise linear approximation of the PM index function. For each PM function, we generated breakpoints in such a way that the difference between the approximate and real function values at any point is less than a certain relative error ϵ of the real function value, i.e. $(\tilde{f}(x) - f(x))/f(x) \leq \epsilon$, where $\tilde{f}(x)$ is the approximate function value and $f(x)$ is the real function value at point x. We tested two levels of ϵ, 0.01 and 0.05. We set $P = 8$, $C = 6$, and $\gamma = 0.05$, and the number of child schedules to be generated by fine tuning is 2.

In figure 10 we first give the average total completion time values for the schedules achieved in different steps of the S_{FUSII} algorithm. Figure 10 shows that, on average, the branching process (Step 3.2) achieves a 45% improvement in the total completion time objective compared with the initial solution (Step 1). For the final schedule achieved in Step 6, the average improvement with respect to the initial schedule reaches 55%.

Figure 11 highlights the importance of proposing an integrated approach with controllable processing times. In most studies in the literature, the processing times are assumed to be fixed. If we want to minimize the processing time effect on the total completion time objective, then we should set $p_i = p^l$ for each job i and schedule them with respect to the SPT rule. On the other hand, in order to minimize the PM effect on the total completion time objective, we should set $p_i = p^u$ for each job i. Controllable processing times provide flexibility in finding solutions with better

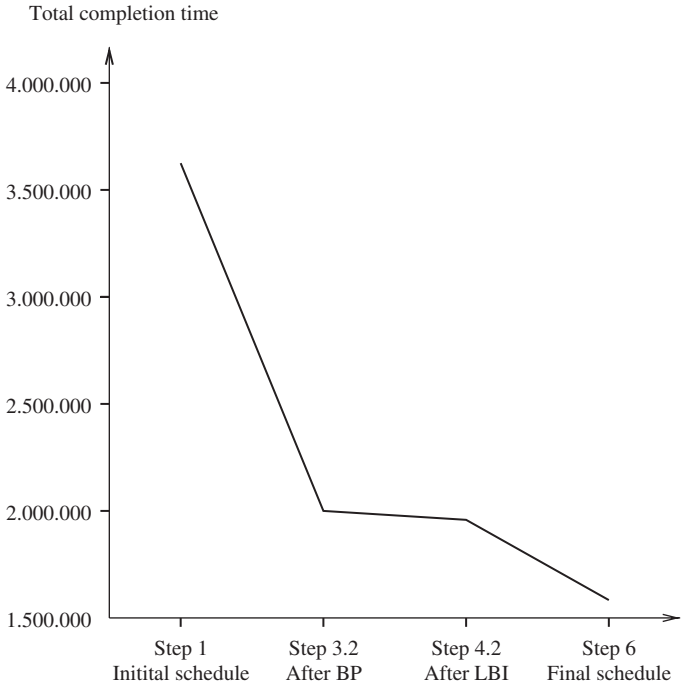

Figure 10. Total completion time at different steps of the S_{FUSII} algorithm.

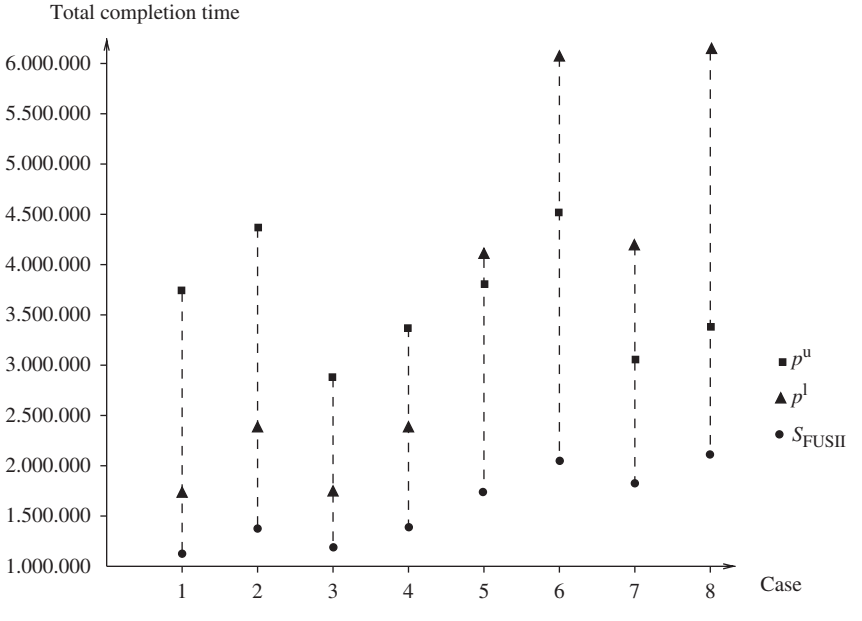

Figure 11. Performance of the S_{FUSII} algorithm.

overall objective function values. To demonstrate the value of controllable processing times, we first solved all of the test problems with fixed processing times (p^l and p^u) and with the proposed S_{FUSII} algorithm, as shown in figure 11. We first compare the schedules found using fixed p^l or p^u. Using fixed p^l instead of p^u gives better performance when T_{PM} is at a low level (Cases 1 to 4), and p^u performs better when T_{PM} is at a high level (Cases 5 to 8), as expected. We can see that the proposed search algorithm gives significantly better total completion time values than the others for all 2^3 experimental settings. This is because our search algorithm finds solutions where the two terms in the total completion time objective are balanced to minimize the overall objective.

In table 2 we give the total completion times and the CPU times (in seconds) achieved using five different versions of the search algorithm for two different levels of the relative error ϵ. The results show that allowing a relative error level of 0.05 instead of 0.01 results in an almost 50% gain in CPU time but only a 0.2% loss in the objective function. We also observe that the total completion time values achieved by the different algorithms are similar. However, the computational effort required by each method is different. The relationship between the CPU time requirements of the different search algorithms is $S_{FUSI} \leq S_{FUSIII} \leq S_{FUSII} \leq S_{TCI} \leq S_{LPB}$. S_{LPB} has the maximum CPU time requirement since it calls the LP solver each time it solves a PM block subproblem. Compared with FUS-based methods, S_{TCI} requires additional computation time for computing the TCI indices of the jobs. Similarly, S_{FUSII} and S_{FUSIII} call the LP solver, which requires additional computation time compared with S_{FUSI}.

Although the average total completion time values achieved by the different versions of the search algorithm are similar, we applied a paired *t*-test to the completion time values of the 80 randomly generated runs found by these algorithms to check the statistical significance of their differences, as given in table 3. The results show that S_{FUSII} outperforms all other versions of the search algorithm. Also, we see that S_{FUSIII} performs the worst. We do not observe a statistically significant difference between S_{FUSI}, S_{TCI} and S_{LPB}. In table 4 we summarize the number of times the minimum total completion time value of a certain algorithm outperforms the others for different T_{PM} levels. Note that more than one algorithm can have the

Table 2. Results of the algorithms for different relative error levels.

		Objective			CPU seconds		
Relative error	Algorithm	Min.	Max.	Average	Min.	Max.	Average
0.01	S_{LPB}	1 148 440	2 162 737	1 605 280	92.90	139.88	114.73
	S_{TCI}	1 148 232	2 163 108	1 605 236	35.63	121.16	69.23
	S_{FUSI}	1 148 234	2 163 133	1 605 236	8.07	21.10	13.07
	S_{FUSII}	1 148 151	2 162 724	1 604 698	11.39	27.11	17.13
	S_{FUSIII}	1 148 152	2 169 241	1 606 404	9.7	22.98	14.68
0.05	S_{LPB}	1 149 300	2 172 240	1 608 711	51.25	67.67	59.06
	S_{TCI}	1 149 531	2 173 199	1 608 437	20.64	59.58	35.86
	S_{FUSI}	1 149 540	2 173 199	1 608 415	5.43	10.96	7.45
	S_{FUSII}	1 149 211	2 172 240	1 607 827	6.71	13.72	9.28
	S_{FUSIII}	1 149 539	2 173 199	1 609 740	5.48	10.95	7.58

Table 3. Paired *t*-test results.

Pairs	Mean	SD	95 CI for mean		Sig.
			Lower	Upper	
S_{FUSI}, S_{FUSII}	285.2	205.2	239.5	330.9	0.0
S_{TCI}, S_{FUSII}	538.0	1442.2	217.1	859.0	0.001
S_{LPB}, S_{FUSII}	582.0	1658.0	213.0	951.0	0.002
S_{FUSIII}, S_{FUSII}	1705.6	2478.8	1154.0	2257.2	0.0
S_{TCI}, S_{FUSI}	252.8	1451.8	−70.3	575.9	0.123
S_{LPB}, S_{FUSI}	296.8	1652.7	−71.0	664.6	0.112
S_{LPB}, S_{TCI}	44.0	886.9	−153.4	241.4	0.659
S_{FUSIII}, S_{FUSI}	1420.4	2394.7	887.5	1953.3	0.0
S_{FUSIII}, S_{TCI}	1167.6	2578.4	593.8	1741.4	0.0
S_{FUSIII}, S_{LPB}	1123.6	2442.0	580.1	1667.0	0.0

Table 4. Number of best results achieved by each algorithm.

Algorithm	Number of best results		Total
	$T_{PM} = 30$	$T_{PM} = 90$	
S_{FUSII}	30	38	68
S_{FUSI}	0	0	0
S_{LPB}	11	3	14
S_{TCI}	0	1	1
S_{FUSIII}	6	0	6

best value for a certain run if there is a tie. We see that S_{FUSII} achieved the best solution in 68 of 80 problems. S_{LPB} achieved the best result 14 times, 11 of which was when T_{PM} was at a low level.

In this section we have implemented our search algorithm on large sized (2000 jobs) instances of the problem. The results show that our search algorithm can generate high solution quality in a very short computation time. The algorithm can solve larger or smaller instances using appropriate algorithm parameters such as α, the number of parent schedules (P) and the number of child schedules (C).

6. Concluding remarks

This paper deals with a scheduling problem that integrates process planning, PM planning and scheduling decisions to minimize the total completion time of a set of jobs. We employ a PM planning approach which, different from the current literature, considers the impact of the operating conditions (i.e. controllable processing times) on the maintenance requirements such that applying shorter processing times (or a higher production rate) requires more frequent PM visits. We show that there is a nonlinear relationship between the maintenance requirement of a machine and the selected processing times. We provide optimality properties for this problem and we propose a heuristic search algorithm based on these properties.

The proposed algorithm can be used to determine the processing times, the job and PM scheduling decisions simultaneously. We also propose alternative improvement and subproblem solution methods to be used in the search algorithm. The computational results show that the proposed algorithms can achieve a high solution quality by balancing the processing time effect and the PM effect terms in the objective function. Even for the 2000-job case, the algorithm requires quite a small computational effort.

References

Adiri, I., Bruno, J., Frostig, E. and Rinnoay Kan, A.H.G., AHGR, Single-machine flow-time scheduling with a single breakdown. *Acta Informatica*, 1989, **26**, 679–696.

Akturk, M.S., Ghosh, J.B. and Gunes, E.D., Scheduling with tool changes to minimize total completion time: basic results and SPT performance. *Eur. J. Oper. Res.*, 2004, **157**, 784–790.

Akturk, M.S. and Gurel, S., Machining conditions-based preventive maintenance. *Int. J. Prod. Res.*, 2007, **45**, 1725–1743.

Bazaraa, M.S., Sherali, H.D. and Shetty, C.M., *Nonlinear Programming: Theory and Algorithms*, 1993 (Wiley: New York).

Cassady, C.R. and Kutanoglu, E., Minimizing job tardiness using integrated preventive maintenance planning and production scheduling. *IIE Trans.*, 2003, **35**, 503–513.

Cassady, C.R. and Kutanoglu, E., Integrating preventive maintenance planning and production scheduling for a single machine. *IEEE Trans. Reliabil.*, 2005, **54**, 304–309.

Graves, G.H. and Lee, C.Y., Scheduling maintenance and semi-resumable jobs on a single machine. *Nav. Res. Logist.*, 1999, **46**, 845–863.

Hoogeveen, H., Multicriteria scheduling. *Eur. J. Oper. Res.*, 2005, **167**, 592–623.

Jeong, I.J., Leon, V.J. and Villalobos, J.R., Integrated decision-support system for diagnosis, maintenance planning, and scheduling of manufacturing systems. *Int. J. Prod. Res.*, 2006, **45**, 267–285.

Kayan, R.K. and Akturk, M.S., A new bounding mechanism for the CNC machine scheduling problems with controllable processing times. *Eur. J. Oper. Res.*, 2005, **167**, 624–643.

Koomsap, P., Shaikh, N.I. and Prabhu, V.V., Integrated process control and condition-based maintenance scheduler for distributed manufacturing control systems. *Int. J. Prod. Res.*, 2005, **43**, 1625–1641.

Lee, C.Y., Machine scheduling with availability constraints. *Handbook of Scheduling*, pp. 1–22, 2004 (CRC Press: USA).

Lee, C.Y. and Leon, V.J., Machine scheduling with a rate modifying activity. *Eur. J. Oper. Res.*, 2001, **128**, 119–128.

Lee, C.Y. and Liman, S.D., Single machine flow-time scheduling with scheduled maintenance. *Acta Informatica*, 1992, **29**, 375–382.

Ow, P.S. and Morton, T.E., Filtered beam search in scheduling. *Int. J. Prod. Res.*, 1988, **26**, 35–62.

Qi, X., A note on worst-case performance of heuristics for maintenance scheduling problems. *Discr. Appl. Math.*, 2007, **155**, 416–422.

Qi, X., Chen, T. and Tu, F., Scheduling the maintenance on a single machine. *J. Oper. Res. Soc.*, 1999, **50**, 1071–1078.