Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 183 (2007) 591–607
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Discrete Optimization

Optimal allocation and processing time decisions on non-identical parallel CNC machines: ϵ-constraint approach

Sinan Gurel, M. Selim Akturk *

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

Received 16 February 2006; accepted 10 October 2006
Available online 13 December 2006

Abstract

When the processing times of jobs are controllable, selected processing times affect both the manufacturing cost and the scheduling performance. A well known example for such a case that this paper specifically deals with is the turning operation on a CNC machine. Manufacturing cost of a turning operation is a nonlinear convex function of its processing time. In this paper, we deal with making optimal machine-job assignments and processing time decisions so as to minimize total manufacturing cost while the makespan being upper bounded by a known value, denoted as ϵ-constraint approach for a bicriteria problem. We then give optimality properties for the resulting single criterion problem. We provide alternative methods to compute cost lower bounds for partial schedules, which are used in developing an exact (branch and bound) algorithm. For the cases where the exact algorithm is not efficient in terms of computation time, we present a recovering beam search algorithm equipped with an improvement search procedure. In order to find improving search directions, the improvement search algorithm uses the proposed cost bounding properties. Computational results show that our lower bounding methods in branch and bound algorithm achieve a significant reduction in the search tree size that we need to traverse. Also, our recovering beam search and improvement search heuristics achieve solutions within 1% of the optimum on the average while they spent much less computational effort than the exact algorithm.

Keywords: Scheduling; Controllable processing times; Manufacturing cost; Makespan; Non-identical parallel machines

1. Introduction

Most of the studies in the machine scheduling literature assume fixed processing times. However, many industry applications allow us to control the processing times. A well known example is the turning operation on CNC turning machines. On a CNC turning machine, we can control the processing time of an operation by setting the machining parameters such as the cutting speed and feed rate. Decreasing the processing time of a job usually requires incurring extra costs. For a turning operation, decreasing the processing time by increasing the cutting speed and/or feed rate results in more wear on the tool which implies increased tooling cost for the operation. In order to utilize the processing time controllability on a machine, we need to make appropriate processing time decisions which take this time/cost

* Corresponding author. Tel.: +90 312 290 1360; fax: +90 312 266 4054.
E-mail address: akturk@bilkent.edu.tr (M.S. Akturk).

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2006.10.024

trade-off into account. In this study, we try to handle a situation where we need to make two decisions at the same time, which are scheduling jobs on non-identical parallel CNC turning machines and making appropriate processing time decisions, so as to minimize total manufacturing cost. In practice, when the workload on a machine is high, the time objective (makespan) becomes important and we need to consider manufacturing cost and makespan objectives at the same time. One of the methods to solve bicriteria problems in the literature is representing one of the objectives as a constraint and optimizing over the second objective. By this way, we can search over the different values to generate a set of discrete efficient points to approximate the efficient frontier. Therefore, we consider the problem of minimizing total manufacturing cost objective for a given upper limit on the makespan objective. This method known as the ϵ-constraint approach as discussed in T'kindt and Billaut (2006) has been used widely in the literature, because it is easy to use in an interactive algorithm. Moreover, the decision maker can interactively specify and modify the bounds and analyze the influence of these modifications on the final solution. For the resulting single criterion problem, we provide optimality properties that will be used in an exact solution method. We also propose a beam search method, and develop an improvement search algorithm for the problem.

In the current literature on the loading and scheduling problems of flexible manufacturing systems, the most popular performance measure is balancing the workload (or minimizing the makespan). This is due to the fact that these systems require a very high investment cost so that the managers would like to fully utilize their capacity. Unfortunately, the processing times are also assumed as fixed in this literature, although the existing CNC machine technology allows us to change the processing times very quickly by just changing few lines in the CNC programming code. This study considers both the makespan and total manufacturing cost objectives at the same time for a flexible machining environment and gives several methods to find efficient solutions.

Trade-offs between cutting parameters and manufacturing cost or surface quality of a turning operation have been studied for a long time in the literature. Hitomi (1979) included mathematical models and solution methods for different objectives of machine parameter selection problem for the turning operation. Lamond and Sodhi (1997) considered the cutting speed selection and tool loading decisions on a single cutting machine so as to minimize total processing time. Sodhi et al. (2001) considered determining the optimal processing speeds, tool loading and part allocations on several flexible machines with finite capacity tool magazines where the objective is to minimize the makespan. Kayan and Akturk (2005) recently provided a mechanism to determine upper and lower bounds for the processing time of a turning operation. They also showed that manufacturing cost of a turning operation can be expressed as a function of its processing time. In this study, we consider non-identical parallel CNC turning machines on which we will determine the optimum processing times (equivalently, machining parameters) that will minimize the total manufacturing cost ($\overline{F}$) subject to a makespan $C_{\max}$ constraint.

The pioneering study initiating the controllable processing times in scheduling is by Vickson (1980). He considered the total processing cost and total weighted completion time on a single machine. Karabati and Kouvelis (1997) discuss simultaneous scheduling and optimal processing time decision problem for a multi-product, deterministic flow line operated under a cyclic scheduling approach. They provide a solution method for the processing time decision subproblem first and then develop an iterative procedure to solve both problems at the same time. Controllable processing time issue has been receiving increasing attention in recent studies in the scheduling literature. They usually deal with two objectives at the same time: minimizing processing cost and a scheduling objective. Hoogeveen (2005) includes a recent review of those studies that consider controllable processing times in multi-objective scheduling. A well known paper that deals with processing cost and makespan objectives at the same time on non-identical parallel machines is by Trick (1994). He considered controllable processing times where each job has a linear processing cost function. He considered the problem of minimizing total processing cost subject to makespan (capacity) constraints, and showed the NP-hardness of the problem. In this study, an important difference is that we now have a nonlinear convex manufacturing cost function for each job. Moreover, we propose a branch and bound (B&B) algorithm for the problem. We then give a beam search algorithm which can be implemented for the instances where the B&B algorithm is not

computationally efficient. We also propose an improvement search algorithm which can be used to improve any given feasible schedule for the problem.

For the identical parallel machines case, Jansen and Mastrolilli (2004) considered controllable processing times and gave polynomial time approximation schemes for the problems of minimizing sum of processing cost and makespan, minimizing processing cost subject to makespan constraint and minimizing makespan subject to processing cost constraint. They also provided exact algorithms for the preemptive versions. Mastrolilli (2003) considered the problem of minimizing maximum flow time subject to processing cost constraint on identical parallel machines with release dates given for each job. He showed that preemptive case is polynomially solvable but non-preemptive case is NP-hard. Our results that for the non-identical machine problem also apply for the identical machine problems.

In the next section, we give the problem definition. In Section 3, we define the single machine subproblem and give a solution method for the problem. In Section 4, we prove cost lower bounds for the partial schedules which we will employ in our exact and beam search algorithms. Section 5 discusses a heuristic method which constructs a starting solution for the B&B algorithm. We give the B&B algorithm in Section 6. Then, in Section 7, we discuss the beam search algorithm for the problem. We proposed an improvement search heuristic in Section 8. In Section 9, we provide a recovering procedure that improves the beam search algorithm. We computationally test all these methods in Section 10 and finally give concluding remarks in Section 11.

2. Problem definition

The notation used throughout the paper is as follows:

Parameters

J	set of jobs to be processed
$f_{jm}(p_{jm})$	manufacturing cost function of processing time for job j on machine m
p^{l}_{jm}	processing time lower bound for job j on machine m
p^{u}_{jm}	processing time level at which $f_{jm}(p_{jm})$ takes its minimum value
C_m	operating cost for CNC turning machine m (\$/minute)
H_m	maximum applicable cutting power by CNC turning machine m (hp)
T_j, e_j	tooling cost multiplier and exponent for job j

Decision variables

p_{jm}	processing time of job j on machine m
X_{jm}	decision variable, that controls if job j is assigned to machine m

In set J, we have N jobs to be processed. Each job corresponds to a metal cutting (turning) operation that will be performed by a given cutting tool on one of the M CNC turning machines. Each job is different in terms of its length and diameter, depth of cut, maximum allowable surface roughness, and its cutting tool type. Also, each machine is different in terms of its maximum applicable cutting power H_m, and its unit operating cost, C_m (\$/minute). Each job must be performed on a single machine without preemption and each machine can perform one job at a time. We also assume that setup and tool change times are negligible. The most commonly used manufacturing cost function for a turning operation is the sum of operating cost and tooling cost. Operating cost of job j on machine m is the cost of running the machine for its duration p_{jm} and can be expressed as $C_m \times p_{jm}$. Tool usage for a turning operation can be calculated by the ratio of its processing time to its Taylor's tool life formula. Tool cost for turning operation can be calculated by the cost of the tool used times tool usage of the operation. Kayan and Akturk (2005) recently showed that manufacturing cost of a turning operation can be expressed as a function of processing time as follows:

$$f_{jm}(p_{jm}) = C_m \cdot p_{jm} + T_j \cdot (p_{jm})^{e_j}.$$

In this paper, we will be using this form of the manufacturing cost function. Since each job has different manufacturing properties and each machine has a different unit operating cost C_m, manufacturing cost function f_{jm} is different for each job on each machine. Kayan and Akturk (2005) showed that f_{jm} is minimized at a processing time level p^{u}_{jm}. Since f_{jm} is a convex function, it is increasing for $p_{jm} > p^{u}_{jm}$. Then, we take p^{u}_{jm} as an upper bound on p_{jm} since both the manufacturing cost of job j and the makespan for machine m (since it is a regular scheduling measure) increase for $p_{jm} > p^{u}_{jm}$. Furthermore, there exists a processing time lower bound p^{l}_{jm} which is determined by the manufacturing properties of job j and the

maximum applicable cutting power of machine m, therefore p^{l}_{jm} is also different for each job on each machine.

The problem is to schedule N jobs on M machines and find the optimum processing time of each job so as to minimize total manufacturing cost ($\overline{F}$) under the constraint that the makespan ($C_{\max}$) of the schedule being upper bounded by a known value of K. The technique used is the ϵ-constraint approach and is written in the γ field of the $\alpha|\beta|\gamma$ scheduling notation as $\epsilon(\overline{F}/C_{\max})$. Therefore, our bicriteria problem is presented as $R_m|contr|\epsilon(\overline{F}/C_{\max})$ and can be formulated as follows:

$$\min \quad \overline{F}: \sum_j \sum_m X_{jm} \cdot f_{jm}(p_{jm})$$

$$\text{s.t.:} \quad \sum_j X_{jm} \cdot p_{jm} \leqslant K \quad \forall m, \tag{1}$$

$$\sum_m X_{jm} = 1 \quad \forall j, \tag{2}$$

$$p^{l}_{jm} \leqslant p_{jm} \leqslant p^{u}_{jm} \quad \forall j, m, \tag{3}$$

$$X_{jm} \in \{0,1\} \quad \forall j, m. \tag{4}$$

In the mixed integer nonlinear programming (MINLP) model above, the objective function is the total manufacturing cost of the jobs, ($\overline{F}$). Constraint set (1) guarantees that the makespan for each machine is less than or equal to K so that the $C_{\max} \leqslant K$ for the schedule. Constraint set (2) forces each job to be assigned to a machine. Constraint set (3) applies the processing time lower and upper bounds for each job. The objective function is non-convex due to the existing ($X_{jm}f_{jm}(p_{jm})$) terms. Moreover, due to the bilinear terms ($X_{jm}p_{jm}$) in the constraint set (1), the feasible space of the problem is also non-convex, so it is a non-convex MINLP model. The well known Outer Approximation and Generalized Benders Decomposition algorithms for MINLP problems are only valid under the convex objective function and convex feasible space assumptions (Floudas, 1995). The non-convexities constitute the major difficulty in finding the global optimal solutions in MINLP problems and non-convex MINLP problems receive increasing attention in nonlinear optimization theory (Kesavan et al., 2004). In this paper, by exploiting the structure of our non-convex MINLP problem, we give an exact algorithm and propose efficient heuristic algorithms for the problem.

The formulation above shows that we have to make two types of decisions to solve the problem, the first one is machine/job allocation decisions and the second one is processing time decisions. For a given machine/job allocation, we can find the optimal processing times by solving the single machine manufacturing cost minimization problem subject to the makespan constraint for each machine. This is a subproblem of our original problem and we denote it as P_m where m stands for machine m. In the next section we give a solution method for P_m.

3. Single machine subproblem (P_m)

In P_m, we find the optimal processing times for the set of jobs assigned to machine m such that total manufacturing cost is minimized and the makespan for the machine does not exceed K. The problem of finding the optimal processing times on a single machine under a makespan constraint has been studied in the literature. The problem for the case of linear decreasing cost functions is shown to be polynomially solvable by Van Wassenhove and Baker (1982). This result was extended to the case of piecewise linear cost functions by Hoogeveen and Woeginger (2002). Since we have a nonlinear convex cost function, we propose an optimality property and a new algorithm to solve this problem.

Assuming that J_m is the set of jobs assigned to machine m, the single machine problem for machine m can be formulated as follows:

$$(P_m) \quad \min \quad \sum_{j \in J_m} f_{jm}(p_{jm})$$

$$\text{s.t.:} \quad \sum_{j \in J_m} p_{jm} \leqslant K, \tag{5}$$

$$p^{l}_{jm} \leqslant p_{jm} \leqslant p^{u}_{jm} \quad \forall j \in J_m. \tag{6}$$

In Lemma 1, we give a sufficient optimality property for P_m. This property is very important since it states the relationships between processing times of jobs on the same machine in an optimal solution.

Lemma 1 (Optimality property for P_m). *In an optimal solution to the single machine problem P_m, let $p^{\star}$ be the optimal processing times vector, and $J^1_m = \{j : p^{\star}_{jm} > p^{l}_{jm}\}$ and $J^2_m = \{j : p^{\star}_{jm} = p^{l}_{jm}\}$ where $J_m = J^1_m \cup J^2_m$. Then the following conditions holds*:

(i) $(\partial f_{jm}/\partial p_{jm})(p^{\star}_{jm}) = \lambda$ *for* $j \in J^1_m$.
(ii) $(\partial f_{jm}/\partial p_{jm})(p^{\star}_{jm}) \geqslant \lambda$ *for* $j \in J^2_m$.

Proof of Lemma 1. Since makespan is a regular scheduling measure, increasing the completion time of any job will not improve the makespan value.

Consequently, any processing time value greater than p^{u}_{jm} will lead to an inferior solution because both objectives will get worse. Therefore, we can replace $p^{l}_{jm} \leqslant p_{jm} \leqslant p^{u}_{jm}$ in constraint set (6) with $p_{jm} \geqslant p^{l}_{jm}$. Next, we assume that there exists at least one job j on machine m such that $p_{jm} > p^{l}_{jm}$, then the optimal processing times vector $p^{\bigstar}$ is a regular point. Such a point must satisfy the Karush–Kuhn–Tucker (KKT) conditions. Then the Lagrangian function for a processing time vector p is as follows:

$$L(p,\lambda,\mu) = \sum_{j\in J_m} f_{jm}(p_{jm}) - \lambda\left(K - \sum_{j\in J_m} p_{jm}\right) + \sum_{j\in J_m} \mu_j (p^{l}_{jm} - p_{jm}).$$

At the optimal solution $p^{\bigstar}$, $\nabla_p(L_p,\lambda,\mu) = 0$ must be satisfied, then for each $j \in J_m$, the following equation must hold:

$$(\partial f_{jm}/\partial p_{jm})(p^{\bigstar}_{jm}) - \lambda - \mu_j = 0,$$

where $\lambda \leqslant 0$ and $\mu_j \geqslant 0$ for each $j \in J_m$. If $j \in J^{1}_{m}$, then $\mu_j = 0$, so $(\partial f_{jm}/\partial p_{jm})(p^{\bigstar}_{jm}) = \lambda$ which proves part (i) of the lemma. If $j \in J^{2}_{m}$, then $(\partial f_{jm}/\partial p_{jm})(p^{\bigstar}_{jm}) = \lambda + \mu_j$. Since $\mu_j \geqslant 0$, $(\partial f_{jm}/\partial p_{jm})(p^{\bigstar}_{jm}) \geqslant \lambda$ holds, which proves part (ii) of the lemma. The only case that $J^{1}_{m} = \emptyset$ holds for an optimal solution is the case in which $K = \sum_{j\in J_m} p^{l}_{jm}$. Then, the optimal solution is $p^{\bigstar}_{jm} = p^{l}_{jm} \forall j \in J_m$ which falls into the case described in part (ii) of the lemma. □

The expression $(\partial f_{jm}/\partial p_{jm})(p^{\bigstar}_{jm})$ denotes the derivative of the manufacturing cost function f_{jm} with respect to p_{jm} at the point $p^{\bigstar}_{jm}$. The value λ in Lemma 1 gives us the change in the optimal manufacturing cost ($\lambda \times \Delta$), if K is increased by a sufficiently small amount Δ. Property in Lemma 1 defines a local optimal solution to the subproblem (P_m), so that we cannot improve the manufacturing cost by changing the processing times of the jobs, i.e., no improving feasible direction exists. Since the objective function of P_m is a nonlinear convex function and the feasible region is a convex set, a local optimal solution is globally optimal, so that property in Lemma 1 is actually a sufficient optimality property. We know that $\lambda \leqslant 0$ always holds because $f_{jm}(p_{jm})$ is non-increasing in the interval $[p^{l}_{jm}, p^{u}_{jm}]$ for all j and m.

An immediate extension of this property for P_m to non-identical parallel machines case is as follows:

Corollary 1. *For the $R_m|contr|\epsilon(\overline{F}/C_{\max})$ problem, an optimal solution must satisfy the property in Lemma 1 for each machine, individually.*

For the non-identical parallel machines problem, property in Lemma 1 must hold for each machine m individually so we need to solve the subproblem P_m for each machine. The optimality property in Lemma 1 allows us to solve the problem P_m. However, since each job may use a different tool type we may have different e_j's for each job. Then, it is not possible to derive a closed form expression to determine the optimal processing time $p^{\bigstar}_{jm}$. Therefore, in order to solve the problem we need to employ a line search algorithm, which will search over the Lagrangian dual variable of the makespan constraint, λ. According to Lemma 1, for each value of λ we can determine a corresponding processing time for each job and a corresponding makespan level. Then, given a makespan level K, we can search over possible values of λ to achieve the makespan level K and corresponding optimal processing times.

P_m algorithm:

Step 1: Set the upper bound for λ, $\lambda^{u} = 0$.

Step 2: Set the lower bound for λ, $\lambda_l = \min_j\{(\partial f_{jm}/\partial p_{jm})(p^{l}_{jm})\}$.

Step 3: Calculate corresponding makespan levels $K^{u} = \sum_{j\in J_m} p^{u}_{jm}$ and $K^{l} = \sum_{j\in J_m} p^{l}_{jm}$.

Step 4: If $K^{l} > K$, then the problem is infeasible. Else if $K^{u} \leqslant K$, then $p^{\bigstar}_{jm} = p^{u}_{jm}$ for all $j \in J_m$. Else, apply the bisection method (Bazaraa et al., 1993) over all possible values of λ in $[\lambda^{l}, \lambda^{u}]$ in order to find a solution which has a makespan level within ϵ-neighborhood of K.

P_m algorithm is pseudo-polynomial since it applies the bisection method. We will use P_m algorithm in the B&B algorithm, beam search, and improvement search heuristics. In the next section, we will prove useful properties on cost lower bounds for partial schedules. Optimality property in Lemma 1 will also be useful for proving these properties.

4. Cost lower bounds for a partial schedule

In a partial schedule, denoted as S_p, a subset of jobs (J^{p}) is assigned to machines, but the remaining jobs are not yet assigned. For S_p, we denote the set of jobs assigned to machine m as J^{p}_{m} and assume that optimal p_{jm} decisions were made by solving the P_m

for the currently scheduled jobs on machine m. Considering an arbitrary S_p, we will prove lower bounds for the manufacturing costs of complete schedules achievable by adding the unscheduled jobs to S_p. We assume that when we add an unscheduled job to S_p, processing times of previously scheduled jobs may change, but the machine/job assignments in S_p stay the same. Since the processing time decisions for the jobs in J^p were made previously by solving the subproblem P_m for each machine m, we have at hand the optimal dual price λ_m for each machine m. When we add an unscheduled job j to a machine m of the partial schedule S_p, it is sure that the new schedule will have a higher manufacturing cost. Assume that adding an unscheduled job j to machine m does not violate the makespan constraint (1), i.e., $\sum_{i\in J_{pm}} p^{l}_{im} + p^{l}_{jm} \leqslant K$. Then, we give a lower bound for the cost increase that will occur by adding a single job j to a machine m of S_p in Lemma 2.

Lemma 2 (Cost lower bound for adding job j to machine m (lb_{jm})). *A lower bound, lb_{jm}, on the cost change that occurs by adding job j to machine m can be found as follows*:

$$lb_{jm} = f_{jm}(p^{ub}_{jm}) - \lambda_m \cdot p^{ub}_{jm},$$

$$\text{where } p^{ub}_{jm} = \max(p^{l}_{jm}, (\partial f_{jm}/\partial p_{jm})^{-1}(\lambda_m)).$$

Proof. Suppose that we first set the processing time of job j to $p^{ub}_{jm} = \max(p^{l}_{jm}, (\partial f_{jm}/\partial p_{jm})^{-1}(\lambda_m))$ and then add the job to machine m. p^{ub}_{jm} is the processing time that satisfies the optimality property (Lemma 1) for machine m. It is obvious that optimal processing time $p^{\star}_{jm}$, to be achieved after solving P_m, cannot be higher than p^{ub}_{jm}, therefore we will definitely incur an additional cost of at least $f_{jm}(p^{ub}_{jm})$. The new schedule on machine m may be infeasible since the makespan of the jobs on machine m, $\sum_{i\in J^p_m} p_{im} + p^{ub}_{jm}$, may exceed K. In such a case, the jobs on machine m must be compressed to make the schedule feasible. We will estimate the cost of compressing the jobs on machine m to K. The marginal cost of decreasing the makespan of the schedule is $-\lambda_m$ and we need to compress the jobs by p^{ub}_{jm}. Then, the second term is $-\lambda_m \cdot p^{ub}_{jm}$ which is a lower bound on the compression cost to be incurred. Then, the additional cost of the new schedule achieved will be at least lb_{jm}. □

Lemma 2 gives us a lower bound for the cost of adding an unscheduled job to a specified machine. We want to determine a cost change lower bound for adding all unscheduled jobs to S_p. By using the lb_{jm} values for $j \in J\backslash J^p$, we can formulate an integer program (IP) that gives us a lower bound on the manufacturing cost increase due to adding all unscheduled jobs in $J\backslash J^p$ to S_p. A lower bound for the cost increase to be caused by adding all unscheduled jobs, or equivalently forming a complete schedule, can be found by solving the following integer program:

$$\text{(IP)} \quad \min \quad \sum_{j\in J\backslash J^p} \sum_{m=1}^{M} X_{jm} lb_{jm}$$

$$\text{s.t.:} \quad \sum_{j\in J\backslash J^p} X_{jm} p^{l}_{jm} \leqslant K - \sum_{j\in J_{pm}} p^{l}_{jm} \quad \forall m, \quad (7)$$

$$\sum_{m=1}^{M} X_{jm} = 1 \quad j \in J \setminus J^p, \quad (8)$$

$$X_{jm} \in \{0,1\} \quad j \in J \setminus J^p \text{ and } \forall m. \quad (9)$$

In the IP model above, the objective function is the sum of the cost change lower bounds (lb_{jm}) for the possible assignments of unscheduled jobs to the machines. Constraint set (7) is the makespan constraint that guarantees that sum of the processing time lower bounds of jobs assigned to a machine does not exceed K. Constraint set (8) assigns each unscheduled job to a machine. Finally, there exists binary constraints (9) for the decision variable X_{jm}. In the following lemma, we prove that the IP model gives a lower bound for S_p.

Lemma 3 (LB_{IP}). *For a partial schedule S_p, an optimal solution of the* IP *gives a lower bound for the cost increase in order to form a complete schedule.*

Proof. As discussed in Lemma 2, lb_{jm} gives a cost increase lower bound of adding job j to machine m of S_p. The IP model looks for a feasible machine/unscheduled job assignment that gives the minimum sum of lb_{jm}'s, given that each unscheduled job is assigned to a machine and makespan constraint is satisfied for each machine. A complete schedule achievable from S_p will obviously have one of the feasible machine/job assignments of the IP model. Then, forming any complete schedule will bring more cost than the optimal value of the IP model. □

We denote the lower bound found by solving the IP model as LB_{IP}. If an IP for S_p turns out to be infeasible, this means no complete schedule can be achieved from S_p. Next property gives the computational complexity of the IP problem.

Lemma 4 (NP-hardness). *Solving the IP model is NP-hard.*

Proof. As our IP model is looking for an optimal machine/job assignment subject to makespan constraints, it is a Generalized Assignment Problem (GAP) model. Then, finding a cost increase lower bound by solving IP is NP-hard. □

Since we will need to find a cost increase lower bound for each partial schedule (node) in our B&B algorithm, solving an NP-hard problem for each node, of course, may not be efficient in terms of computation time. Therefore, we propose two other practical methods. The first one is to solve the LP relaxation of the IP model. The following lemma is obvious:

Lemma 5 (LB_{LP}). *An optimal solution of the* LP *relaxation of* IP *gives a cost increase lower bound for* S_p.

A lower bound (LB_{LP}) to be found by the LP relaxation is obviously smaller than LB_{IP}. However, it requires much less computation time which is critical for our B&B algorithm. Another approach to find a lower bound for S_p would be relaxing the makespan constraint (7) of IP. In such a case we still get a lower bound for the cost increase, which is denoted as LB_{R}, and given below:

Lemma 6 (LB_{R}). *A lower bound for the cost increase required to form a complete schedule from* S_p *is given by*: $LB_R = \sum_{j \in J \setminus J^p} \min_m lb_{jm}$, *where* lb_{jm} *is the cost increase lower bound of adding job* j *to machine* m *of* S_p *as defined in Lemma 2.*

Proof. The lower bound on the additional cost to be incurred by adding a job j to S_p (without knowing the machine to which it will be assigned) can be found by $\min_m lb_{jm}$. When we have multiple unscheduled jobs, we can find an overall lower bound by summing the corresponding lower bounds for all unscheduled jobs. □

The lower bound LB_{R} assumes that each job can be assigned to the machine that gives the best cost change lower bound for it regardless of the makespan. Obviously, such a machine/job assignment may be infeasible due to constraint set (7), but computing LB_{R} is much more simpler than LB_{IP} or LB_{LP}. Between three given lower bounding methods, it is easy to see the relationship given below:

Lemma 7. $LB_{\mathrm{R}} \leqslant LB_{\mathrm{LP}} \leqslant LB_{\mathrm{IP}}$.

Computational requirements will also have the same relationship. In order to achieve a better lower bound we need to solve a harder problem. In this section, we proposed three methods to find a cost increase lower bound for an arbitrary partial schedule S_p. In the next section, we describe a construction heuristic to find an initial solution for the B&B algorithm.

5. Initial solution

In order to find an initial solution for the problem, we propose a heuristic algorithm denoted as IS. This initial solution will serve as an upper bounding solution for our B&B algorithm. The IS algorithm starts with a list of jobs where jobs were ordered in ascending order of their minimum cost ($\min_m f_{jm}(p^{\mathrm{u}}_{jm})$). Then, starting with the first job in the list and all machines being empty at the beginning, in each iteration, the algorithm adds a new job to the schedule. For each job, the algorithm first selects a suitable machine which gives the minimum cost increase lower bound as discussed in Lemma 2, and then adds the job to that machine by solving the subproblem P_m for the machine.

IS algorithm

Step 1: List the jobs in ascending order of their $\min_m f_{jm}(p^{\mathrm{u}}_{jm})$.

Step 2: Starting from the first job in the list, for each job do Steps 3 to 5.

Step 3: Calculate lb_{jm} for each machine m, and choose the best machine: $m' = \mathrm{argmin}_m lb_{jm}$.

Step 4: Check the feasibility of assigning job j to machine m'. If it is not feasible, choose the next best machine and update m'. Repeat this until finding a suitable machine or finding out that no machine is feasible. If no feasible machine exists, then stop.

Step 5: Assign job j to machine m' and determine the optimal processing times by solving the single machine subproblem P_m for machine m'.

IS algorithm schedules the minimum cost job first, so it is a greedy approach in a sense. At each iteration, a new job is scheduled on the machine which gives the minimum cost increase lower bound given in Lemma 2. IS algorithm either ends with a feasible schedule for the problem or fails to find a feasible schedule and stops. The algorithm performs

at most $N \times M$ iterations. In the next section, we will give the B&B algorithm.

6. B&B algorithm

The major difficulty in designing a B&B algorithm for a non-convex MINLP problem is computing a lower bound at a node of a B&B tree. As discussed in Kesavan et al. (2004), in few studies that exist in the area the main approach is constructing the convex hull of the non-convex feasible region and convex envelope of the non-convex objective function, and solving the resulting convex MINLP model to achieve a lower bound. Differently, in this paper we used the properties of partial solutions and proposed three lower bounding methods that are easy to compute and implement in practice. In this section, we first explain the B&B search tree. Then, we discuss the node elimination rules. We next give a step-by-step description of the algorithm.

6.1. Search tree

At the root node of the search tree at level 0, all jobs are unscheduled. Each node in the search tree corresponds to an assignment where the jobs in a subset of J are assigned to the machines. At each level of the search tree, the B&B algorithm assigns an unscheduled job. Then, a node at level k corresponds to a partial schedule with k jobs being assigned to the machines and similarly a node at level N corresponds to a complete schedule where all jobs in J are scheduled.

The algorithm uses a job list $(j_1,\ldots,j_N)$ to assign job j_k in the kth level of the tree. The root node has M child nodes: one distinct node for scheduling job j_1 to each machine m for $m = 1,\ldots,M$. Then, each node at level 1 corresponds to an assignment where job j_1 is assigned on a different machine. Similarly, a node at level k corresponds to a partial schedule with jobs $(j_1,\ldots,j_k)$ assigned on the machines. Each node at level $k < N$ has at most M child nodes so that there is one child node for the assignment of job j_{k+1} to machine m, for $m = 1,\ldots,M$.

For each node, we find the optimal cost and the optimal processing time decisions for each machine by solving the subproblem P_m for the given machine/job assignments of the partial schedule. This will allow us to use the lower bounding methods discussed in Lemmas 2–6 so that we will be able to reduce the tree size by fathoming some parts of the tree. Obviously, by traversing the search tree defined above, we can find an optimal solution for the problem.

6.2. Node elimination

Having a search tree that enumerates all possible solutions for the problem and finds an optimal schedule, the question is how to reduce the size of the search tree by discarding nodes. There are two ways of eliminating nodes from our B&B tree. One is by feasibility and the other is by optimality. As discussed above, we generate a child node from a parent node by adding a new job j to a machine m of the partial schedule represented by the parent node. When opening a child node, we first check if it is feasible to add job j to machine m. If the child node is not feasible ($\sum_{i\in J_m} p^{\mathrm{l}}_{im} + p^{\mathrm{l}}_{jm} > K$), we eliminate the child node, so that ignore all subtree growing from that child node.

If a child node turns out to be feasible, then we solve the single machine subproblem P_m for machine m and make optimal processing time decisions for the partial schedule so that Lemma 1 is satisfied for the jobs on machine m. After checking the feasibility of the node and solving the subproblem for machine m, the next step is to find a lower bound for the new partial schedule. In Section 4, we derived three different ways of calculating manufacturing cost increase lower bounds for achieving complete schedules from a given partial schedule. We can employ one of those methods ($LB_{\mathrm{IP}}, LB_{\mathrm{LP}}, LB_{\mathrm{R}}$) to find a cost increase lower bound. The cost increase lower bound of the node plus the cost of the partial schedule of the node itself gives us a lower bound for a complete schedule achievable from the node. If the lower bound of a node is higher than or equal to the cost upper bound, then the node is eliminated due to optimality.

Another alternative for eliminating nodes by feasibility is detecting the infeasibility of a partial schedule when finding its lower bound by using LB_{IP} or LB_{LP}. When solving the IP model (or its LP relaxation) for finding LB_{IP} (LB_{LP}), the solver may find out that the model is infeasible. This shows that no feasible complete schedule can be achieved from the considered partial schedule. Then, we eliminate the node by feasibility. Having described the search tree and the ways of eliminating nodes from the tree we give a stepwise presentation of our B&B algorithm below:

B&B algorithm

Step 1: Find an initial upper bounding solution by using the IS algorithm. Set upper bounding cost UB^c to the cost of the solution found by IS. If IS cannot find a feasible solution, set $UB^c = \infty$.

Step 2: Form a list of jobs $(j_1, \ldots, j_N)$ in descending order of $(\max_m p^1_{jm})$.

Step 3: Start with the root node as the parent node, set the level of the parent node $level_p$ to 0.

Step 4: For $i = 1, \ldots, m$, do the following:

Step 4.1: Generate child node i of the parent node by adding job j_{level_p+1} to machine i.

Step 4.2: Check the feasibility of child node i. If child node i is not feasible, eliminate it.

Step 4.3: Else, solve the subproblem P_m for machine i and calculate cost (F^p) of the partial schedule.

Step 4.4: If child node i is a complete schedule, i.e. $level_p + 1 = N$, then, if $F^p < UB^c$, set $UB^c = F^p$. Else, apply Steps 4.5 and 4.6.

Step 4.5: Find the cost increase lower bound LB^p. If it turns out that no complete feasible schedule can be achieved from child node i, then eliminate it. Else, calculate lower bound for child node i by $LB^C = F^p + LB^p$.

Step 4.6: If $LB^c \geqslant UB^c$ eliminate child node i.

Step 5: Find the next parent node and update $level_p$ and go to Step 4. If no parent node is available, then stop.

In the B&B algorithm, Step 1 finds an initial schedule to be an upper bounding solution by using the IS algorithm. In Step 2, jobs are ordered to form a list which determines which job to be scheduled at what level of the B&B search tree. Step 3 sets the root node as the first parent node to be considered in the following steps. Step 4 and its sub-steps branches on the parent node and generates its child nodes. At each child node, job j_{level_p+1} is added to a different machine of the partial schedule represented by the parent node. Step 4.2 checks the feasibility of adding job j_{level_p+1} to machine i and if not feasible eliminates the node. If it turns out to be feasible, in Step 4.3, single machine subproblem P_m is solved for machine i. If child node i represents a complete schedule and if $F^p < UB^c$, then the schedule on child node i is the best solution found so far and the UB^c value is updated. If child node i represents a partial schedule, a lower bound is calculated for the partial schedule in Step 4.5. At this step, we may conclude that no feasible complete solution can be achieved from this partial schedule, then we eliminate this node. In Step 4.6, if the lower bound found in Step 4.5 is greater than UB^c, then we eliminate child node i. In Step 5, we either find a new parent node or stop.

In the B&B algorithm, we implemented a modified depth-first strategy. When we branch on a parent node, we generate all its child nodes and out of these child nodes we select the one with the minimum lower bound as the new parent node and branch on this node next. If the complete subtree growing from the selected parent node is traversed, we branch on the next best child node of its parent node. This is a depth-first strategy supported with a greedy node selection approach.

The B&B algorithm defined above either ends up with an optimal solution or concludes that the problem is infeasible. The performance of this B&B algorithm is bounded by the computational requirements. In the next section we propose a beam search algorithm for the instances where applying B&B is inefficient.

7. Beam search algorithm (BS)

Up to now we have described an exact algorithm (B&B) for the problem. We have also proposed lower bounding methods to reduce the tree size for this algorithm. However, the problem is an NP-hard problem and the size of the search tree for the B&B algorithm increases exponentially as N and M increase. For higher levels of N, M and K, we propose a beam search algorithm to find near optimal solutions. Beam search is a fast B&B method which keeps best b (beam width) nodes at a level of search tree and eliminates the rest. Therefore, its running time is polynomial in the problem size. A well known beam search application on a scheduling problem is by Ow and Morton (1988). They applied beam search on single machine early/tardy problem. The performance of a beam search method depends on the quality of the method that the algorithm uses to select the best nodes at a level of the tree. In our BS algorithm, we will consider the same search tree structure as our B&B algorithm. Therefore, each node at a higher level than N will correspond to a partial schedule and at each level a new job from a list of jobs will be scheduled. As a rule to choose the nodes to be saved, we will use the lower bound LB_{LP}. Each lower bounding method we defined in Section 4, can be used as an evaluation method

which estimates the cost of complete schedule for a given partial schedule. A stepwise description of the beam search algorithm for the problem is below:

BS algorithm

Step 1: Form a list of jobs $(j_1,\ldots,j_N)$ in descending order of $(\max_m p^1_{jm})$.

Step 2: Start with the root node as the parent node. Set the level of the parent node $level_p$ to 0.

Step 3: For each selected parent node and for $i = 1,\ldots,m$, do Step 3.1 to 3.4:

Step 3.1: Generate child node i of the parent node by adding job j_{level_p+1} to machine i.

Step 3.2: Check the feasibility of child node i. If child node i is not feasible, eliminate it.

Step 3.3: Else, solve the subproblem P_m for machine i and calculate cost of the partial schedule F^p.

Step 3.4: If $level_p < (N-1)$, then find the cost change lower bound LB^p. If it turns out that no complete feasible schedule can be achieved from child node i, then eliminate it. Else, calculate the lower bound for node i by $LB^C = F^p + LB^p$.

Step 4: If all child nodes are eliminated due to feasibility, then no feasible solution could be found, stop.

Step 5: If $level_p < (N-1)$, then select best b child nodes with smallest LB^C values, set $level_p = level_p + 1$ and go to Step 2.

Step 6: If $level_p = (N-1)$, then select the best child node with minimum cost and stop.

If the problem is feasible, the BS algorithm either cannot find a feasible solution (Step 4) or finds an approximate optimal solution (Step 6). If the problem is infeasible, the BS algorithm cannot find a feasible solution (Step 4). Using our search tree structure that we proposed in Section 6 and our lower bounding methods as an evaluation function, the BS algorithm is a fast alternative for the cases that the B&B algorithm fails to undertake. The time complexity of the BS algorithm is O(mnb). In the next section, we will propose improvement search steps for the problem which can be applied to any given schedule.

8. Improvement search heuristic (ISH)

We have given an exact algorithm (B&B) for the problem and then proposed a beam search method that runs in polynomial time. In this section, we extend our discussion to an improvement search algorithm. We will define an improvement search heuristic which starts with an initial schedule and improves the solution at each iteration to achieve a local optimal solution.

Our improvement search heuristic starts with an initial schedule which satisfies the optimality condition in Corollary 1 so that we assume the single machine subproblem is solved for each machine. We represent such a solution as a partition of the jobs to the machines. We define two moves to describe the neighborhood of a solution. The first one is 1-move, which is to move a job j from its current machine m_1 to another machine m_2. The other one is 2-swap, which is to exchange job j_1 on machine m_1 with another job j_2 on machine m_2. Given a solution, we proved cost change lower bound properties for the two moves we defined above. Lemma 8 gives the cost change lower bound for a 1-move:

Lemma 8 (Lower bound for a 1-move). *Given a schedule which satisfies the condition in Corollary 1, assume that job j has a processing time p_{jm_1} on machine m_1 and machines m_1 and m_2 have the dual price values λ_{m_1} and λ_{m_2}, respectively. Then, a lower bound for the cost change that will result by moving job j from machine m_1 to m_2 is as below*

$$LB(j:(m_1 \to m_2)) = \lambda_{m_1} p_{jm_1} - f_{jm_1}(p_{jm_1}) + f_{jm_2}(p^{ub}_{jm_2}) - \lambda_{m_2} p^{ub}_{jm_2},$$

where $p^{ub}_{jm_2} = \max((\partial f_{jm_2}/\partial p_{jm_2})^{-1}(\lambda_{m_2}), p^1_{jm_2})$.

Proof. Suppose that we first remove job j from machine m_1, the cost change lower bound for this action is the first two terms of the lower bound expression above. The first term is the cost of job j on machine m_1 and the second one is the lower bound for cost change to occur by expanding the processing times of the remaining jobs on the machine. Suppose that we next add job j to machine m_2. The cost change lower bound for adding job j to m_2 can be calculated as discussed in Lemma 2. The cost change lower bound for this action is third and fourth terms of the lower bound expression. □

Lemma 8 can help us to decide to make a 1-move or not. Since $LB(\Delta_{1\text{-move}})$ is a lower bound, if it is a positive value for a particular 1-move, then it is sure that the move will make the cost objective worse, so

we ignore the move since it is non-improving. Else, if it is negative, then it promises a reduction in cost but since it is a lower bound it does not guarantee a reduction. Hence, we need to find out the actual improvement. Next, we analyze the cost change lower bound for 2-swap moves. Lemma 9 gives a lower bound for the resulting cost change for this move.

Lemma 9 (Lower bound for a 2-swap). *Given a schedule which satisfies the condition in Corollary 1, assume that jobs j_1 and j_2 have processing times $p_{j_1m_1}$ and $p_{j_2m_2}$ and scheduled on machines m_1 and m_2, respectively. Machines m_1 and m_2 have the dual prices λ_{m_1} and λ_{m_2}, respectively. Then, a lower bound for the cost change that will result by moving job j_1 from machine m_1 to m_2 and job j_2 in opposite way is as below*

$$\begin{aligned} LB(j_1 \leftrightarrow j_2) &= \lambda_{m_1}(p_{j_1m_1} - p^{ub}_{j_2m_1}) - f_{j_1m_1}(p_{j_1m_1}) + f_{j_2m_1}(p^{ub}_{j_2m_1}) \\ &\quad + \lambda_{m_2}(p_{j_2m_2} - p^{ub}_{j_1m_2}) - f_{j_2m_2}(p_{j_2m_2}) + f_{j_1m_2}(p^{ub}_{j_1m_2}), \end{aligned}$$

where $p^{ub}_{j_1m_2} = \max((\partial f_{j_1m_2}/\partial p_{j_1m_2})^{-1}(\lambda_{m_2}), p^{1}_{j_1m_2})$ *and* $p^{ub}_{j_2m_1} = \max((\partial f_{j_2m_1}/\partial p_{j_2m_1})^{-1}(\lambda_{m_1}), p^{1}_{j_2m_1})$.

Proof. The proof can be easily done by using Lemma 8. □

If the cost change lower bound for a move is non-negative, then it is sure that the move cannot improve the cost. If it is negative, we call the move as a "promising" move. A promising move may improve the cost, but since we just have a negative lower bound for the cost change, the real cost change after implementing the move may still be positive. Using this fact the proposed algorithm could only evaluate the promising moves which will make it computationally more efficient than a local search algorithm that tries all possible moves. Moreover, the lower bounds presented in Lemmas 8 and 9 will guide any search algorithm to try the most promising move first, like a steepest descent algorithm in some sense. Given an initial schedule, by calculating the cost change lower bounds for all possible 1-moves and 2-swaps, we can either conclude that the schedule is locally optimal, which is the case when all lower bounds are non-negative, or we can try the moves which promise possible improvements since they have negative cost change lower bounds. By using these observations, we will propose an improvement search heuristic for the problem.

The improvement search heuristic starts with an initial schedule. First, the heuristic uses promising 1-moves to improve the initial schedule. To do this, it generates all possible 1-moves for this schedule and calculates the cost change lower bound for each possible 1-move. The heuristic applies the most promising move first and solves the single machine subproblems for the affected machines. If an improvement is achieved, new moves are generated for the new schedule. If no improvement is achieved by this move, the heuristic tries the next most promising move, until an improvement is achieved or no promising move is left. When no improvement is possible for the current schedule by using 1-moves, the heuristic considers 2-swap moves. It tries to improve the solution by 2-swap moves in the same way as we did by 1-moves and stops when no improvement is possible.

Improvement search heuristic (ISH):

Step 1: Take an initial schedule (S) and its cost is $F(S)$.
Step 2: Generate all 1-moves for S and calculate $LB(j:(m_1 \rightarrow m_2))$ for each 1-move.
Step 3: If no promising moves exist, go to Step 5. Else, find the most promising move.
Step 4: Apply the selected move on S and solve the P_m subproblem for the affected machines. The new solution is $F(S')$.
Step 4.1: If the solution is improved, replace S with S' and go to Step 2.
Step 4.2.: Else, find the next most promising move and go to Step 4. If no promising move is found go to Step 5.
Step 5: Generate all 2-swap moves for S and calculate $LB(j_1 \leftrightarrow j_2)$ for each 2-swap move.
Step 6: If no promising moves exist, terminate. Else, find the most promising move and go to Step 7.
Step 7: Apply the selected move on S and solve the P_m subproblem for the affected machines. The new solution is $F(S')$.
Step 7.1: If the solution is improved replace S with S' and go to Step 5.
Step 7.2: Else, find the next most promising move and go to Step 7. If no promising 2-swap move is found, terminate.

By using 2-swap moves defined for ISH, in the next section we will extend the BS algorithm to a recovering beam search algorithm.

9. Recovering beam search (RBS)

Recovering beam search method, as described in detail by Della Croce et al. (2004), combines the beam search idea with local search techniques to improve the performance of a classical beam search technique. The main idea in the recovering step is to prevent the elimination of promising nodes (nodes that could lead to optimal or near optimal solutions) due to errors in the node evaluation step of beam search algorithms by applying local search techniques to achieve better partial solutions at each level of the beam search algorithm. A recovering beam search implementation for the single machine total completion time problem with release dates is given in Della Croce and T'kindt (2002).

In Step 3 of the proposed BS algorithm, we generate child nodes for a given level of the search tree. Assuming that K child nodes generated at level l, our recovering step is as follows:

Recovering procedure

Step 1: Sort the child nodes in non-decreasing order of their LB^C. Let n_k be the kth best node.

Step 2: Define the set of selected nodes, empty set $S = \{\}$.

Step 3: Set $k = 1$. While ($|S| < b$) and ($k < K$) do

Step 3.1: Generate new partial schedules (nodes) by swapping last added job j_{l+1} with each job added to a different machine.

Step 3.2: If a node ($n_k^\star$) with smaller LB^C is found, and if $n_k^\star \notin S$ then $S = S \cup \{n_k^\star\}$. Else $S = S \cup \{n_k\}$.

We put this recovering procedure into Step 5 of BS algorithm as below:

Step 5: If $level_p < (N-1)$, then apply Recovering Procedure to select b nodes, set $level_p = level_p + 1$ and go to Step 2.

The time complexity of the RBS algorithm is $O(mn^2b)$. In the next section, we give the results of our computational study.

10. Computational results

In this paper, we first developed an exact algorithm (B&B) for the problem with three different lower bounding methods. We next proposed a beam search (BS) algorithm along with an improvement search heuristic (ISH). We further extended BS to a recovering beam search (RBS) algorithm. We coded these algorithms in C language and compiled with Gnu C compiler version 2.95.3. All codes were run on the operating system Solaris 2.7 on a workstation Sun HPC 4500 with 12×400 MHz UltraSPARC CPU and 3GB memory. The B&B, BS and RBS algorithms used the CPLEX 9.1 commercial solver to compute the lower bounds LB_{LP} and LB_{IP}. All reported computational times are in seconds.

We considered two experimental factors: number of jobs ($N = 10, 15, 20$) and number of machines ($M = 2, 3, 4$). For each experimental setting we took 5 replications. For each replication we generated cutting specifications (diameter, length, depth of cut and required surface roughness) of jobs randomly. For each job we randomly used one of the tool types out of ten types of cutting tools with different technical coefficients given in Kayan and Akturk (2005). We randomly generated the cost of each tool C_t from the uniform distribution $U \sim [5, 10]$. C_t determines the cost coefficient T_j with other job and tool specific parameters as discussed in Kayan and Akturk (2005). We used four types of machines with the following C_m, H_m couples: $(0.3, 5)$, $(0.5, 10)$, $(0.7, 15)$ and $(0.9, 20)$. The CNC machines with higher horsepower, H_m, capabilities can attain higher cutting speeds and feed rates (i.e. lower processing times), but their initial investment cost (and their operating cost) would be higher as well. This way we can evaluate the impact of different CNC machine technologies on the scheduling decisions. In our computational runs, when $M = 2$, we used first two machines described above and when $M = 3$, we used first three machines.

Another very important factor for the problem is the limit (K) on makespan objective of the schedule. How to select a K value for a given problem setting is a critical decision since selecting a very small K value may cause all instances of a replication to be infeasible. In order to see the effect of K, we solved each replication of the problem for 5 different levels of K. To find proper K values, we first solved the makespan minimization problem for each replication for fixed processing times case where $p_{jm} = p'_{jm}$ for each j and m. This is a makespan minimization problem on unrelated machines and known to be NP-hard, so we used a polynomial-time algorithm by Davis and Jaffe (1981), which was shown to have a worst case bound of $(1 + \sqrt{2})\sqrt{M}$. This algorithm provides us a feasible makespan level K which we denote as K_{DJ}. We cal-

culated five different K levels by using the formula $K = k \times K_{DJ}$ where $k = 0.6, 0.8, 1, 1.2, 1.4$. In the computational study, we first solved randomly generated problems by using three different B&B algorithms. Each B&B algorithm uses a different lower bounding method that we proposed in Section 4. The B&B algorithm either finds out that a problem is infeasible or gives us an optimal solution. For the cases that B&B found an optimal solution we solved the problem by the BS and RBS algorithms which use LB_{LP} as a partial schedule evaluation tool. We next tested ISH algorithm using 3 different starting solutions provided by BS, RBS and IS algorithms.

A critical step in our B&B algorithm is deciding the job order (Step 2), i.e. determining which job to be added to a partial schedule next at a given level of the search tree. We ordered the jobs in a descending order of $\max_m p^l_{jm}$. This is intuitive because as we schedule the jobs with higher processing time lower bound at earlier stages of the B&B tree, we catch infeasible schedules earlier and this reduces the number of nodes to be opened and decreases the computation time. In computational study, we considered two other rules to order jobs in Step 2 of B&B and took trial runs to compare different methods. One method is to order the jobs in ascending order of $\min_m f_{jm}(p^u_{jm})$ as in the IS algorithm, which allows us to schedule lower cost jobs earlier in the B&B tree. We also considered $\max_m f_{jm}(p^u_{jm})$, which allows us to schedule highest minimum cost jobs earlier. We took trial runs for $N = 10$ and $M = 2$ and 3. We give the average results for CPU, number of opened nodes, number of eliminated nodes due to feasibility and number of eliminated nodes by lower bound (optimality) in Table 1. The results show that our selection of $\max_m p^l_{jm}$ order performs better than the other ordering rules both for the CPU requirement and for the node elimination capability due to feasibility and optimality.

Table 1
Trial results for job ordering rules for Step 2 of B&B

Job order	CPU	Opened nodes	Eliminated due to feasibility	Eliminated by lower bound
$\max_m p^l_{jm}$	0.20	1426	29,199	10,923
$\min_m f_{jm}(p^u_{jm})$	1.85	16,056	21,941	3550
$\max_m f_{jm}(p^u_{jm})$	1.45	12,171	25,333	4043

We next discuss the performance of the B&B algorithm with different lower bounding methods. We consider the cases where a feasible solution is available for the problem. We give the size of the eliminated B&B tree and traversed nodes for different lower bounding methods in Table 2. For a given (N, M) instance, the maximal number of nodes to be traversed in worst case in our B&B tree can be calculated by $(1 - M^{N+1})/(1 - M)$. For $M = 4$ and $N = 20$, total number of nodes to be traversed may reach 1,466,015,503,701. Similarly, when we decide to fathom a node at level L, we save from opening $(1 - M^{N-L+1})/(1 - M) - 1$ nodes which is the number of nodes that would grow from the fathomed node at the worst case. We measured the number of eliminated nodes due to bounds and feasibility in terms of their percentages to the maximal total number of nodes. The results in Table 2 show that our lower bounds can reduce the tree size by 29% on the average. There are instances where this reduction reaches to 87%. The feasibility effect reduced the tree size by 66% on the average. The last column of the table for traversed tree size shows that we could solve the problems by just opening 4.6% of the nodes on the average. There are cases solved by just traversing a negligible size of B&B tree. We have shown in Section 4 in Lemma 7 that for a given partial schedule (node) the following relationship holds: $LB_{IP} \geqslant LB_{LP} \geqslant LB_R$. We also observe this relationship between the sizes of the eliminated B&B trees by different lower bounding methods in Table 2.

In Table 3, we present the CPU requirements of different lower bounding methods in B&B algorithm for different experimental settings. This table shows that increasing N or M strongly affects the running

Table 2
Eliminated and traversed tree sizes

LB type	Eliminated by lower bound			Eliminated due to feasibility			Traversed		
	Mean (%)	Min (%)	Max (%)	Mean (%)	Min (%)	Max (%)	Mean (%)	Min (%)	Max (%)
LB_R	26	0	86.4	68.4	0.4	100	5.6	0	43
LB_{LP}	28.8	0	87.2	66.5	0.4	100	4.7	0	38.4
LB_{IP}	29.1	0	87.2	66.3	0.4	100	4.6	0	38.2

Table 3
CPU requirements (in seconds) for different lower bounding methods

N	LB type	$M=2$			$M=3$			$M=4$		
		Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
10	LB_R	0.08	0.02	0.16	0.26	0.01	0.59	1.18	0.08	2.30
	LB_{LP}	0.26	0.10	0.46	1.11	0.05	2.33	5.03	0.44	10.51
	LB_{IP}	0.32	0.14	0.52	1.42	0.05	2.89	6.11	0.25	10.6
15	LB_R	2.06	0.20	5.81	21.3	0.26	80.6	241	0.99	1002
	LB_{LP}	4.88	0.69	12.7	65.4	1.21	268	662	3.67	3422
	LB_{IP}	5.99	1.74	13.5	90.7	0.28	314	927	2.90	4196
20	LB_R	56.2	3.84	186	2443	33.2	7380	69,950	236	177,030
	LB_{LP}	112	7.93	382	4853	114	18,266	101,293	618	362,148
	LB_{IP}	129	22.2	386	7550	25.9	24,468	169,676	319	584,398

time of the B&B algorithm as expected. For the considered N and M levels, LB_R has the minimum average running time. The second best alternative is the LB_{LP} in terms of the CPU time. An important observation in Table 3 is that as N and M increase the CPU time required by LB_R approaches to the CPU time required by LB_{LP}, so we may expect to see that LB_{LP} will have shorter CPU times for larger problem sizes. If we check the CPU time ratio LB_R/LB_{IP}, we observe that as N is increased, performance of the LB_R gets closer to the performance of LB_{IP}, but as M is increased, we observe the opposite. This is due to the fact that computing LB_{IP} is itself an NP-hard problem and requires much more time when M is increased. Another observation in Table 3 is that for each lower bounding method we see big gaps between minimum and maximum CPU times. This is because we solve each problem for different K levels as discussed below.

Table 4 gives the average size of the eliminated and traversed nodes and required CPU time for $N=20$ and $M=4$ for different K levels, such that $K=k\times K_{DJ}$ where $k=0.6,0.8,1,1.2,1.4$. For example, $k=0.6$ corresponds to the case where K level is 1 and so on. We observe that as K is increased, the size of the traversed tree increases since fewer number of nodes are eliminated due to the feasibility. Hence, the CPU time required to solve the problem increases, too. We see that the CPU time requirement when K level is 5 is twenty times higher than the CPU time requirement when K level is 3. This shows that CPU requirement of the B&B algorithm is strongly affected by K. Therefore, we can say that the B&B algorithm is more efficient for smaller K's in terms of running time.

Table 4
Eliminated and traversed nodes at different K levels for $N=20$ and $M=4$ by LB_{LP}

K level	Eliminated by lower bound (%)	Eliminated by feasibility (%)	Traversed (%)	CPU
3	1.4	98.6%	0.0%	10,402
4	7.5%	92.5%	0.004%	93,921
5	18.7%	81.2%	0.014%	199,556

In Table 5, we give the solution quality results for the proposed IS, BS and RBS algorithms. We use a beam width $b=3$ for BS and RBS. We define the relative solution quality of an algorithm A, R_A, as the ratio of the difference between cost achieved by A and the optimal cost achieved by B&B over the optimal cost expressed in %. It is the percentage deviation from the optimum. The average performance of IS algorithm varies between 7.2% and 22.9%. BS algorithm achieves an average performance between 1.8% and 7.9%. RBS algorithm gives the best results with an average performance between 0.1% and 1.4%. There are cases where BS and RBS achieve the optimum. The worst performance for RBS is 9.6% whereas it is 26.5% for BS. We observe that including a recovering step in BS algorithm significantly improved the solution quality. When we check the CPU time performance for each heuristic, we see that all three methods are very efficient. As an example, for the largest problem size of the (20, 4) case, the average CPU time requirements were 0.28 and 0.71 CPU seconds for the BS and RBS algorithms, respectively. We also observe that the IS algorithm has negligible CPU time requirements.

In Table 6, we give the solution quality results for the ISH algorithm for three different starting solutions provided by IS, BS and RBS. We represent the deviation of ISH from the optimum as R_{A+ISH} where A stands for the algorithm of which is used

Table 5
Deviations from the optimum for IS, BS and RBS algorithms

N	M	R_{IS} (%)			R_{BS} (%)			R_{RBS} (%)		
		Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
10	2	11	0.6	43.1	1.8	0	11.5	0.1	0	0.7
	3	22.6	2.8	45.7	4.8	0	24.6	0.6	0	3.9
	4	22.9	6.1	40.6	2.1	0.1	11.9	1.4	0	9.6
15	2	7.4	0.4	24.5	3.7	0	21.5	0.4	0	2.6
	3	18.4	8.7	27.8	5.4	0.1	16.9	0.5	0	2.6
	4	15.4	10.0	22.3	5.0	0.4	23.5	0.8	0.1	3
20	2	7.2	0.1	21.5	5.0	0	19.9	0.4	0	1.6
	3	14.3	4.6	28.7	7.9	0.5	26.5	0.9	0	4.3
	4	17.2	8.5	28.9	5.3	0.6	17.4	1.1	0	5.6

Table 6
Deviations from the optimum for ISH algorithm

N	M	R_{IS+ISH} (%)			R_{BS+ISH} (%)			$R_{RBS+ISH}$ (%)		
		Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
10	2	0.5	0	3.9	1.1	0	11.5	0.06	0	0.7
	3	2.4	0	6.5	4.3	0	23.1	0.5	0	3.8
	4	1.9	0	7.1	1.7	0	11.9	0.1	0	9.3
15	2	0.3	0	4.5	1.8	0	12.4	0.1	0	2.2
	3	1.0	0	3.1	3.2	0	11.8	0.4	0	2.6
	4	1.5	0	3.2	4.4	0	23.2	0.5	0	2.8
20	2	0.4	0	3.4	2.5	0	19.9	0.1	0	0.4
	3	1.5	0	4.6	4.9	0.2	20.9	0.7	0	4.3
	4	1.6	0	3.2	4.4	0.4	16.5	0.9	0	5.4

as a starting solution by the ISH algorithm. In comparison with the starting solutions, we observe that the ISH achieves a significant improvement in all three cases. When we compare the results given in Table 5 with respect to Table 6, the average gap between the IS algorithm and the optimum solution was 22.9% (R_{IS}) for the (10, 4) case in Table 5. After implementing the ISH algorithm over the IS algorithm, it was decreased to 1.9% (R_{IS+ISH}) for the (10, 4) case in Table 6. In all cases, the average improvements over the starting solutions were statistically significant.

We next analyzed the performances of BS and ISH algorithms for different K levels as reported in Table 7. A very important observation is that solution quality of BS and ISH algorithms improve as K is increased. Therefore, for the problem instances where our B&B algorithm is not computationally efficient, our BS and ISH algorithms can achieve solutions more closer to the optimum. This is due to the shape of the manufacturing cost function. When K is increased, we deal with higher processing time values where the manufacturing cost functions are flatter.

Table 7
Performances of beam search and improvement search heuristics at different K levels

K level	R_{BS} (%)				R_{ISH} (%)			
	Mean	Min	Max	Standard deviation	Mean	Min	Max	Standard deviation
2	9	0	21.5	7	5.6	0	19.9	6
3	6.2	0	26.5	7	4.4	0	23.2	6
4	3	0	18.2	5	2.3	0	18.2	4
5	2.1	0	12.4	3	1.2	0	8.7	2

Table 8
Relative performances of RBS and ISH with respect to IS

N	M	I_{RBS} (%)			$I_{RBS+ISH}$ (%)			CPU_{RBS}			$CPU_{RBS+ISH}$		
		Mean	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
50	2	4.6	0.1	14.9	5	0.2	15.1	4.16	3.40	4.56	2.63	0.01	10.56
	3	11.5	5.3	25.7	11.7	5.4	26.4	5.73	5.45	6.21	1.13	0.12	4.34
	4	9.0	7.0	11.7	9.4	7.4	12.2	7.39	6.84	8.48	2.65	1.19	4.13
100	2	4.2	−0.1	13.5	4.6	0.2	13.6	27.79	23.25	29.80	50.85	0.13	194.21
	3	9.8	4.7	16.1	10.1	4.9	17.0	42.30	39.60	44.91	22.49	1.10	64.95
	4	9.6	6.5	17.3	10.0	6.6	18.3	56.69	52.22	62.76	27.92	13.67	57.12

Finally, we tested IS, RBS and ISH algorithms for 50–100 jobs and 2, 3, 4 machines. We cannot solve these instances to the optimum due to the CPU time requirements. Therefore, we compared the results achieved by the RBS algorithm with respect to the initial results given by the IS algorithm in Table 8. I_{RBS} is the percentage deviation of RBS from the initial solution achieved by IS. We observe that for 50-job problems, the average improvement of the RBS algorithm over the IS is 8.4%. When we apply the RBS together with ISH, the average improvement, $I_{RBS+ISH}$, becomes 8.7%. The required CPU times are still reasonable even for the large problem instances. In Table 8, $CPU_{RBS+ISH}$ indicates the additional CPU time requirement of the ISH algorithm over the RBS algorithm.

11. Conclusion

In this paper, we considered the problem of minimizing total manufacturing costs on non-identical parallel CNC machines with an upper limit on the makespan of the schedule. We provided an exact algorithm (B&B) for the problem along with three alternative lower bounding methods. To the best of our knowledge, our algorithm is the first exact algorithm for this problem. We further proposed a recovering beam search algorithm which employs our lower bounding methods as an evaluation function for partial schedules. Finally, we gave an improvement search algorithm for the problem. For this algorithm, we showed two properties which provide improving search moves for a given schedule. Our computational results show that the proposed exact algorithm can solve the problems by just traversing the 5% of the maximal possible B&B tree size and the proposed lower bounding methods can eliminate up to 80% of the search tree. For the cases where B&B is not computationally efficient, our beam search and improvement search algorithms achieved solutions within 1% of the optimum on the average in a very short computation time. As a future research, we would like to extend this study to include the tool change times in the makespan objective and consider the tool loading decisions for finite capacity tool magazines.

Acknowledgements

The authors would like to thank two anonymous referees whose constructive comments have been used to improve this paper.

References

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

Davis, E., Jaffe, J.M., 1981. Algorithms for scheduling tasks on unrelated processors. Journal of the Association for Computing Machinery 28, 721–736.

Della Croce, F., T'kindt, V., 2002. A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem. Journal of the Operational Research Society 53, 1275–1280.

Della Croce, F., Ghirardi, M., Tadei, R., 2004. Recovering beam search: Enhancing the beam search approach for combinatorial optimization problems. Journal of Heuristics 10, 89–104.

Floudas, C.A., 1995. Nonlinear and Mixed-Integer Optimization. Oxford University Press, New York.

Hitomi, K., 1979. Manufacturing Systems Engineering: A Unified Approach to Manufacturing Technology and Production Management. Taylor and Francis, London.

Hoogeveen, H., 2005. Multicriteria scheduling. European Journal of Operational Research 167, 592–623.

Hoogeveen, H., Woeginger, G.J., 2002. Some comments on sequencing with controllable processing times. Computing 68, 181–192.

Jansen, K., Mastrolilli, M., 2004. Approximation schemes for parallel machine scheduling problems with controllable processing times. Computers and Operations Research 31, 1565–1581.

Karabati, S., Kouvelis, P., 1997. Flow-line scheduling problem with controllable processing times. IIE Transactions 29, 1–14.

Kayan, R.K., Akturk, M.S., 2005. A new bounding mechanism for the CNC machine scheduling problems with controllable processing times. European Journal of Operational Research 167, 624–643.

Kesavan, P., Allgor, R.J., Gatzke, E.P., Barton, P.I., 2004. Outer approximation algorithms for separable non-convex mixed-integer nonlinear programs. Mathematical Programming 100, 517–535.

Lamond, B.F., Sodhi, M.S., 1997. Using tool life models to minimize processing time on a flexible machine. IIE Transactions 29, 611–621.

Mastrolilli, M., 2003. Notes on max flow time minimization with controllable processing times. Computing 71, 375–386.

Ow, P.S., Morton, T.E., 1988. Filtered beam search in scheduling. International Journal of Production Research 26, 35–62.

Sodhi, M.S., Lamond, B.F., Gautier, A., Noël, M., 2001. Heuristics for determining economic processing rates in a flexible manufacturing system. European Journal of Operational Research 129, 105–115.

T'kindt, V., Billaut, J.-C., 2006. Multicriteria Scheduling: Theory, Models and Algorithms, second ed. Springer, Berlin.

Trick, M.A., 1994. Scheduling multiple variable-speed machines. Operations Research 42, 234–248.

Van Wassenhove, L.N., Baker, K.R., 1982. A bicriterion approach to time/cost tradeoffs in sequencing. European Journal of Operational Research 11, 48–54.

Vickson, R.G., 1980. Two single-machine sequencing problems involving controllable job processing times. AIEE Transactions 12, 258–262.