Computers & Operations Research 37 (2010) 779 – 789

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

Bicriteria robotic operation allocation in a flexible manufacturing cell

Hakan Gultekin[a], M. Selim Akturk[b,*], Oya Ekin Karasan[b]

[a]Department of Industrial Engineering, TOBB-University of Economics and Technology, 06560 Sogutozu, Ankara, Turkey
[b]Department of Industrial Engineering, Bilkent University, 06800 Bilkent, Ankara, Turkey

ARTICLE INFO

Available online 8 July 2009

Keywords:
Robotic cell
Bicriteria scheduling
CNC
Operation allocation

ABSTRACT

Consider a manufacturing cell of two identical CNC machines and a material handling robot. Identical parts requesting the completion of a number of operations are to be produced in a cyclic scheduling environment through a flow shop type setting. The existing studies in the literature overlook the flexibility of the CNC machines by assuming that both the allocation of the operations to the machines as well as their respective processing times are fixed. Consequently, the provided results may be either suboptimal or valid under unnecessarily limiting assumptions for a flexible manufacturing cell. The allocations of the operations to the two machines and the processing time of an operation on a machine can be changed by altering the machining conditions of that machine such as the speed and the feed rate in a CNC turning machine. Such flexibilities constitute the point of origin of the current study. The allocation of the operations to the machines and the machining conditions of the machines affect the processing times which, in turn, affect the cycle time. On the other hand, the machining conditions also affect the manufacturing cost. This study is the first to consider a bicriteria model which determines the allocation of the operations to the machines, the processing times of the operations on the machines, and the robot move sequence that jointly minimize the cycle time and the total manufacturing cost. We provide algorithms for the two 1-unit cycles and test their efficiency in terms of the solution quality and the computation time by a wide range of experiments on varying design parameters.

1. Introduction

In order to be successful in today's highly competitive market, the companies have to adapt to the environment in which they operate, be more flexible in their operations, and satisfy different market segments. For these purposes Flexible Manufacturing Cells (FMC) are installed and used in most of the manufacturing industries. A manufacturing cell consisting of a number of Computer Numerical Control (CNC) machines and a material handling robot is called a FMC. These systems must be managed successfully to attain the maximum throughput rate with minimum cost. The problem considered in this paper has three aspects: (i) scheduling of robot moves, (ii) determination of the allocation of the operations, and (iii) determination of the optimal processing time of each operation, in a 2-machine cell producing identical parts. In the literature, the most common objective is the maximization of the throughput which is equivalent to the minimization of the cycle time. Although cost objectives are common in scheduling theory and practice and have even higher priority in "process planning", as far as the authors know, there are no studies in robotic cell literature considering cost objectives, except Gultekin et al. [7]. Additionally, in comparison to single criterion approaches, considering multiple criteria provides useful insights for the decision maker. For example, a solution that minimizes the cycle time can be very poor costwise. In this paper, we will consider a bicriteria problem where the objectives are the minimization of the cycle time and the minimization of the total manufacturing cost.

In some industries such as automotive and electronics, due to the material handling and robot work envelope restrictions, a required set of operations must be allocated to a series of stations and each part must go through all the stations in the same sequence. This production environment, generally known as the flow shop type robotic cell, is the most widely studied setting in robotic cell scheduling literature starting with the seminal paper of Sethi et al. [12] and followed by others as summarized in a recent review of Dawande et al. [6]. In this study, we limit ourselves to flexible manufacturing cells in which CNC machines are arranged in a flow shop type production environment. CNC machines are highly flexible, a property that can be readily utilized in improving the productivity of the underlying robotic cells. Furthermore, the machining parameters such as the

* Corresponding author. Fax: +90 312 266 4054.
E-mail address: akturk@bilkent.edu.tr (M. Selim Akturk).

0305-0548/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2009.06.025

speed and the feed rate are controllable variables for such machines. The processing times of the operations on these machines can be changed by altering these parameters. Despite these facts, the current literature assumes the processing time of a part on a machine to be a fixed predetermined parameter and hence limits the number of alternatives. In this study, we assume each of the identical parts to have a set of operations to be performed on the two CNC machines. Each operation requires a specific type of tool and the machines are capable of performing an operation as long as the required cutting tool is loaded on their tool magazines. Consistent with the existing literature, it is assumed that all parts must be processed by both of the machines in the sequence respecting the layout. In other words, we shall decide on the assignment of nonempty sets of operations to each of the machines. Following this decision, the proper cutting tools will be loaded on the machines. In addition to utilizing the flexibility of assigning the operations to the machines, the processing times of the operations on the machines will also be considered as decision variables. Allowing allocation flexibility and controllable processing times in turn affect both the cycle time and the total manufacturing cost. The problem is not only to find the robot move sequence but also to determine the allocation of the operations to the machines and the processing times of the operations on the machines that jointly minimize the total manufacturing cost and the cycle time.

There is an extensive literature on robotic cell scheduling problems with widespread reviews such as Crama et al. [5] and Dawande et al. [6]. A common trend in the existing studies is to consider the minimization of the cycle time as the single objective. Sethi et al. [12] develop the necessary framework for the robotic cell scheduling problems and prove that 1-unit cycles minimize the cycle time for 2-machine cells. An *n-unit cycle* is defined to be a robot move cycle in which all machines are loaded and unloaded exactly n times and the initial and the final states of the cell are the same. Crama and van de Klundert [4] describe a polynomial time dynamic programming algorithm for minimizing the cycle time over all 1-unit cycles in an m-machine cell producing identical parts. Akturk et al. [1] consider a flexible manufacturing cell producing identical parts where the allocations of the operations are decision variables and prove that an n-unit cycle, where $n \leq 2$, minimizes the cycle time. They also provide the regions of optimality for each of the potentially optimal robot move cycles.

Since we have two criteria, the optimal solution will not be unique but instead a set of nondominated solutions will be identified. The reader is referred to Hoogeveen [8] for a review on multicriteria scheduling models. A recent survey of the literature on controllable processing times can be found in Shabtay and Steiner [13]. In the current study, we assume a nonlinear, strictly convex, and differentiable cost function. Although assuming the cost function to be linear simplifies the problem, it is not realistic because it does not reflect the law of diminishing returns. Kayan and Akturk [10] consider a single machine bicriteria scheduling model with controllable processing times. They select total manufacturing cost and any regular scheduling measure—one which cannot be improved by increasing the processing times—such as makespan or cycle time, as the two objective criteria and derive lower and upper bounds on processing times. Gultekin et al. [7] extend this idea to the robotic cell scheduling problem, where the allocations of the operations to the machines are taken as parameters.

The organization of this paper is as follows: In the next section we will present the mathematical formulation of the problem. The solution procedures for the 1-unit cycles will be developed in Sections 3 and 4, respectively. In Section 5, the performance of the proposed heuristic presented in Section 4 will be evaluated through a computational study. Section 6 is devoted to the concluding remarks.

2. Problem formulation

In this study, justified with the complexity of the problem and consistent with most of the studies in the robotic cell scheduling literature, we restrict ourselves to 1-unit cycles. The following robot activity definition borrowed from [4] is sufficient to represent these move cycles in a flow shop type robotic cell:

Definition 1. A_i is the robot activity defined as robot unloads machine i, transfers the part from machine i to machine $(i+1)$, and loads this machine.

In 2-machine cells there are two 1-unit cycles, namely, S_1 cycle with activity sequence $A_0A_1A_2$ and S_2 cycle with activity sequence $A_0A_2A_1$. In the initial state of S_1 cycle, the system is empty and the robot is in front of the input buffer. After the listed activities are performed, the robot returns to the input buffer. Initially in the S_2 cycle, only the second machine is loaded and the robot is in front of the input buffer. The animated views of these cycles can be found at the web site http://www.ie.bilkent.edu.tr/~robot. Let P_i represent the processing time of a part on machine i=1, 2; δ represent the robot transportation time between any two consecutive machines; and ε represent the loading/unloading time of the machines by the robot. The cycle time is defined as the long run average time required to produce one part. Let T_S represent the cycle time of the robot move cycle S. The cycle times of these cycles are provided by Sethi et al. [12] as follows:

$$T_{S_1} = 6\varepsilon + 6\delta + P_1 + P_2, \tag{1}$$

$$T_{S_2} = 6\varepsilon + 8\delta + \max\{0, P_1 - 2\varepsilon - 4\delta, P_2 - 2\varepsilon - 4\delta\}. \tag{2}$$

It is apparent from Eqs. (1) and (2) that the cycle times are sensitive to the processing times P_1 and P_2. We assume each of the identical parts to have a set of operations, $O = \{1, 2, \ldots, p\}$. Processing time of operation l is denoted by t_l. The machines are assumed to be capable of performing any operation so long as they are equipped with the required cutting tools. Akturk et al. [1] prove that by considering the allocation of the operations to the machines as a decision variable, the efficiency of the cells can be improved in terms of the cycle time. The tooling of the machines is done depending on the allocation of the operations. The processing time of a part on a machine is equal to the summation of the processing times of the operations performed by that machine. Let x_{li} be the binary variable that indicates whether operation l is allocated to machine i or not. Then, P_i can be written as $\sum_{l=1}^{p} x_{li}t_l$. The processing times of the operations on the CNC machines are functions of the machining parameters such as the cutting speed and the feed rate. Different parameters yield different processing time values. Kayan and Akturk [10] provide lower and upper bounds for the processing times when minimizing a convex cost function and any regular scheduling measure. Note that the cycle time is a regular scheduling measure. The lower bound of a processing time results from constraints such as the limited tool life, machine power, and surface roughness. On the other hand, the upper bound of a processing time is the value which minimizes the total manufacturing cost. When we analyze the cycle time equations of S_1 and S_2 given in (1) and (2), respectively, it is evident that beyond the cost minimizing processing time value both objectives get worse. Note that the lower bound corresponds to the minimum processing time–maximum cost case whereas the upper bound corresponds to the maximum processing time–minimum cost case. Let t_l^L and t_l^U denote the lower and upper bounds for the processing time of operation l and $f_l(t_l)$ denote the manufacturing cost incurred by the same operation. Since the machines are assumed to be identical, the cost of an operation is machine independent and does not depend on other operation costs. We assume $f_l(t_l)$ to be a

strictly convex and differentiable function. It is also monotonically decreasing for $t_l^L \le t_l \le t_l^U$, $l=1,2,\ldots,p$. The total manufacturing cost incurred by all of the operations can be written as $\sum_{l=1}^{p} f_l(t_l)$. This function is also convex and decreasing for $t_l^L \le t_l \le t_l^U$, $\forall l$. Obviously, the total manufacturing cost does not depend on the robot move cycle but depends only on the processing times of the operations whereas the cycle time depends on both.

In this study, a feasible solution, say ξ, selects either S_1 or S_2 as the robot move cycle, decides on which machine to perform each operation, and determines the processing times of all the operations satisfying their respective bounds. We will consider S_1 and S_2 cycles individually. Let F_1 denote the total manufacturing cost and F_2 denote the cycle time. These are two competing objectives. Hence, there is not a unique optimal solution of this problem but a set of nondominated solutions. In the context of bicriteria optimization theory, solution ξ_1 dominates solution ξ_2 if it is not worse than ξ_2 under any of the performance measures, and is strictly better under at least one of the performance measures. Nondominated solutions are classified as Pareto optimal. More formally:

Definition 2. We say that ξ_1 dominates ξ_2 and denote it as $\xi_1 \preceq \xi_2$ if and only if $F_1(\xi_1) \le F_1(\xi_2)$ and $F_2(\xi_1) \le F_2(\xi_2)$, one of which holds as a strict inequality. A solution ξ^* is called Pareto optimal, if there is no other ξ such that $\xi \preceq \xi^*$. If ξ^* is Pareto optimal, the point $z^* = (F_1(\xi^*), F_2(\xi^*))$ is called efficient or nondominated. The set of all efficient points is the efficient frontier.

There are different ways to deal with bicriteria problems [8]. In this study, we will use the *epsilon-constraint method* denoted by $\varepsilon(f|g)$. Here f and g represent the two performance measures. In this approach, nondominated points are found by solving a series of problems of the form minimize f given an upper bound on g. By this method a finite number of nondominated points is determined. In other words, $\varepsilon(F_1|F_2)$ is solved for a number of specific F_2 values which are used to estimate the entire efficient frontier. Estimating the entire efficient frontier means that the cycle time values for which we solve the epsilon-constraint problem are uniformly spread over the range of all feasible cycle time values. In the following sections, we solve $\varepsilon(F_1|T)$, where T is an upper bound on the cycle time, by considering each one of the cycles individually.

3. Solution procedure for the S_1 cycle

In this section we will develop a solution procedure for the $\varepsilon(F_1|T)$ for the S_1 cycle. It is obvious from Eq. (1) that the cycle time of S_1 does not depend on the allocation of the operations to the machines. Hence, we get $6\varepsilon+6\delta+\sum_{l=1}^{p} t_l \le T$ as the cycle time bound constraint. Letting $\hat{T} = T - 6\varepsilon - 6\delta$ we have the following formulation for the S_1 cycle:

$$\varepsilon(F_1|\hat{T})^{S_1}: \min \quad \sum_{l=1}^{p} f_l(t_l) \tag{3}$$

$$\text{s.t.} \quad \sum_{l=1}^{p} t_l \le \hat{T}, \tag{4}$$

$$t_l \ge t_l^L, \quad \forall l. \tag{5}$$

This formulation minimizes the cost for a given bound on the cycle time. Notice that we eliminated the upper bounds in the formulation above. This is because, these upper bounds are not physical bounds but as mentioned earlier they are calculated from the problem characteristics. The upper bound of a processing time is selected as the processing time value that minimizes its cost function. Let us use $\partial f_l(t_l)$ instead of $\partial f_l(t_l)/\partial t_l$ for notational simplicity. More formally, the upper bounds are calculated using the equation, $\partial f_l(t_l)|_{t_l=t_l^U} = 0$. Since the cost function is convex, beyond this minimizer it is an increasing function that satisfies $\partial f_l(t_l)|_{t_l=\hat{t}_l} > 0$ for $\hat{t}_l > t_l^U$. That is, the cost function is increasing beyond $\hat{t}_l > t_l^U$. Hence, $t_l^* \le t_l^U$ always holds for optimal t_l^* and thus can be eliminated from the above formulation.

This formulation is of the form of a nonlinear knapsack problem with separable, convex continuous objective function and constraints for which different solution approaches are reviewed in [2]. In the sequel, we will develop a problem specific solution procedure for the formulation above. This formulation is also equivalent to a single machine makespan minimization problem with p jobs and controllable processing times. Since $\varepsilon(F_1|\hat{T})^{S_1}$ minimizes a strictly convex function over a convex closed set, a local minimum of F_1 is a global minimum and there exists exactly one global minimum (see [3, Proposition 2.1.1]). Let $\boldsymbol{t}^* = (t_1^*, t_2^*, \ldots, t_p^*)$ be the optimal solution of $\varepsilon(F_1|\hat{T})^{S_1}$ throughout this section. Let $T_{S_1}^L$ and $T_{S_1}^U$ be the lower and upper bounds of the cycle time of the S_1 cycle, respectively. In other words

$$T_{S_1}^L = 6\varepsilon + 6\delta + \sum_{l=1}^{p} t_l^L \quad \text{and} \quad T_{S_1}^U = 6\varepsilon + 6\delta + \sum_{l=1}^{p} t_l^U. \tag{6}$$

Also let $\hat{T}^L = T_{S_1}^L - 6\varepsilon - 6\delta$ and $\hat{T}^U = T_{S_1}^U - 6\varepsilon - 6\delta$. Note that $\varepsilon(F_1|\hat{T})^{S_1}$ is infeasible if the cycle time bound in constraint (4) satisfies $\hat{T} < \hat{T}^L$ and all solutions are dominated if $\hat{T} > \hat{T}^U$. As a result we have the following lemma which will play an essential role in the development of the solution procedure:

Lemma 1. *In the optimal solution of* $\varepsilon(F_1|\hat{T})^{S_1}$ *for* $\hat{T}^L \le \hat{T} \le \hat{T}^U$, *constraint* (4) *is satisfied as equality.*

Proof. Let $F_1^* = \sum_{l=1}^{p} f_l(t_l^*)$ be the optimal objective function value of $\varepsilon(F_1|\hat{T})^{S_1}$ with optimal processing time vector $\boldsymbol{t}^*$. Assume to the contrary that $\sum_{l=1}^{p} t_l^* < \hat{T}$ and consider another solution with, $\hat{t}_l^* = t_l^*$, $\forall l \ne \hat{l}$ for an arbitrary index $\hat{l}$ such that $t_{\hat{l}}^* < t_{\hat{l}}^U$. Let $\hat{t}_{\hat{l}}^* = t_{\hat{l}}^* + \beta$ for some β, $0 < \beta \le \min\{t_{\hat{l}}^U - t_{\hat{l}}^*, \hat{T} - (\sum_{l=1}^{p} t_l^*)\}$. This new solution has identical processing times for all operations except $\hat{l}$ and $\hat{t}_{\hat{l}}^* > t_{\hat{l}}^*$. Since the cost function is decreasing with respect to processing times, the objective function of the new solution, $\hat{F}_1^*$, satisfies $\hat{F}_1^* < F_1^*$. However, this contradicts with $\boldsymbol{t}^*$ being the optimal solution of $\varepsilon(F_1|\hat{T})^{S_1}$. □

As a consequence of the above lemma, we know that the sum of the optimal processing times is equal to the cycle time bound. Consider the partition induced by $\boldsymbol{t}^*$, i.e., $J = \{l : t_l^* > t_l^L\}$ and $\bar{J} = \{h : t_h^* = t_h^L\}$. We know that if $\hat{T} > \hat{T}^L$, then $J \ne \emptyset$. The following result determines the properties of the operations of these two sets.

Lemma 2. *In the optimal solution of* $\varepsilon(F_1|\hat{T})^{S_1}$, *where* $\hat{T} > \hat{T}^L$ *the following conditions hold*:

(i) $\partial f_l(t_l)|_{t_l=t_l^*} = \partial f_k(t_k)|_{t_k=t_k^*}$, $\forall l, k \in J$,
(ii) $\partial f_l(t_l)|_{t_l=t_l^*} \le \partial f_h(t_h)|_{t_h=t_h^*}$, $\forall h \in \bar{J}$ *and* $\forall l \in J$.

Proof. Since we assume $\hat{T} > \hat{T}^L$, there exists at least one l such that $t_l^* > t_l^L$. Therefore, the vector $\boldsymbol{t}^* = (t_1^*, t_2^*, \ldots, t_p^*)$ is a regular point. In other words, the gradients of the active inequality constraints and the gradients of the equality constraints are linearly independent at that point. Such a point must satisfy the Karush–Kuhn–Tucker (KKT) conditions. From Lemma 1, constraint (4) is satisfied as equality. As a consequence, the Lagrangian function for point $\boldsymbol{t}^*$ can be written as

follows:

$$L(t^*,\lambda^*,\mu^*)=\sum_{l=1}^{p} f_l(t_l^*)+\lambda^*\left(\sum_{l=1}^{p} t_l^*-\hat{T}\right)+\sum_{l=1}^{p}\mu_l^*(t_l^L-t_l^*).$$

Here, $\mu_l^*\geq 0$ and λ^* is unrestricted in sign. If we set $\nabla_t(L(t^*,\lambda^*,\mu^*))=0$, we get $\partial f_l(t_l)|_{t_l=t_l^*}+\lambda^*-\mu_l^*=0$, $\forall l$. If $l\in J$, then $\mu_l^*=0$. Thus, $\partial f_l(t_l)|_{t_l=t_l^*}=-\lambda^*$, $\forall l\in J$, which proves (i). On the other hand, if $h\in \bar{J}$, then $\partial f_h(t_h)|_{t_h=t_h^*}=-\lambda^*+\mu_h^*$. Since $\mu_h^*\geq 0$, $\partial f_h(t_h)|_{t_h=t_h^*}\geq -\lambda^*=\partial f_l(t_l)|_{t_l=t_l^*}$, $\forall h\in\bar{J}$ and $\forall l\in J$, which proves (ii). □

Up to now, we know that the operations are partitioned into two sets with respect to their processing time values in the optimal solution to $\varepsilon(F_1|\hat{T})^{S_1}$. Additionally, the lemma above identifies some properties of the elements of these two sets regarding their processing time values. Note that the derivative of the cost function at a processing time value shows the contribution of a small change in the processing time to the cost. Let α_l represent the contribution of operation l when its processing time is at its lower bound. More formally, we can write $\alpha_l=\partial f_l(t_l)|_{t_l=t_l^L}$, $l=1,2,\ldots,p$. For a particular operation, say operation l, we can determine a cycle time value such that beyond this value of the cycle time the processing time of operation l will be greater than its lower bound. This cycle time value is called the critical cycle time value for operation l and denoted as M_l. In order to calculate such a value for operation l, we first calculate processing time values of all the remaining operations that have the same contribution with the lower bound of this particular operation. This can be calculated using $\partial f_h^{-1}(\alpha_l)$, $h\in O$, where ∂f^{-1} represents the inverse of the derivative of the cost function f. However, the processing times cannot be smaller than their lower bounds. In order to satisfy this, we use $\max\{t_h^L,\partial f_h^{-1}(\alpha_l)\}$. Finally, in order to get the critical value, we sum all these values for all operations:

$$M_l=\sum_{h\in O}\max\{t_h^L,\partial f_h^{-1}(\alpha_l)\}. \tag{7}$$

The following lemma uses these values to determine the elements of the J and $\bar{J}$ sets easily without determining the optimal processing times to $\varepsilon(F_1|\hat{T})^{S_1}$.

Lemma 3. *In the optimal solution of $\varepsilon(F_1|\hat{T})^{S_1}$, $l\in J$, if and only if $\hat{T}>M_l$.*

Proof (*Proof by contradiction*). Let us first prove the necessity: assume that $\hat{T}>M_h$ but to the contrary $h\in\bar{J}$, for at least one operation h. Hence, $t_h^*=t_h^L$. But from condition (ii) of Lemma 2, $\alpha_h=\partial f_h(t_h)|_{t_h=t_h^L}\geq\partial f_l(t_l)|_{t_l=t_l^*}$, $\forall l\in J$. Using the convexity and the invertibility of f_l we can get a bound on t_l^* as $t_l^*\leq\partial f_l^{-1}(\alpha_h)$, $\forall l\in J$. Inserting this bound inside $\hat{T}=\sum_{l\in\bar{J}}t_l^L+\sum_{l\in J}t_l^*$, we get the following inequality:

$$\hat{T}=\sum_{l\in\bar{J}}t_l^L+\sum_{l\in J}t_l^*\leq\sum_{l\in\bar{J}}t_l^L+\sum_{l\in J}\partial f_l^{-1}(\alpha_h).$$

Finally, since $\hat{T}>M_h$, we get the following contradiction:

$$\sum_{l=1}^{p}\max\{t_l^L,\partial f_l^{-1}(\alpha_h)\}\geq\hat{T}>\sum_{l=1}^{p}\max\{t_l^L,\partial f_l^{-1}(\alpha_h)\}.$$

Now let us prove the sufficiency: assume $h\in J$ but to the contrary $\hat{T}\leq M_h$, for at least one operation h. Hence, $t_h^*>t_h^L$, which implies $\alpha_h=\partial f_h(t_h)|_{t_h=t_h^L}<\partial f_h(t_h)|_{t_h=t_h^*}$. From condition (i) of Lemma 2, $\partial f_h(t_h)|_{t_h=t_h^*}=\partial f_l(t_l)|_{t_l=t_l^*}$, $\forall h,l\in J$. Hence, we have $\alpha_h=\partial f_h(t_h)|_{t_h=t_h^L}<\partial f_l(t_l)|_{t_l=t_l^*}$, $\forall l\in J$. Since f_l is convex, it satisfies, $t_l^*>\partial f_l^{-1}(\alpha_h)$. Combining this with $\hat{T}=\sum_{l\in\bar{J}}t_l^L+\sum_{l\in J}t_l^*$,we get

$$\hat{T}=\sum_{l\in\bar{J}}t_l^L+\sum_{l\in J}t_l^*>\sum_{l\in\bar{J}}t_l^L+\sum_{l\in J}\partial f_l^{-1}(\alpha_h).$$

Using $\hat{T}\leq M_h$, we reach a contradiction

$$\sum_{l=1}^{p}\max\{t_l^L,\partial f_l^{-1}(\alpha_h)\}<\hat{T}\leq\sum_{l=1}^{p}\max\{t_l^L,\partial f_l^{-1}(\alpha_h)\}.$$ □

This lemma enables a very powerful preprocessing scheme in the solution procedure of $\varepsilon(F_1|\hat{T})^{S_1}$. Clearly, the breakpoint M_l for each operation l can be calculated easily from the given cost functions and processing time lower bounds. A simple comparison of these breakpoints against $\hat{T}$ partitions the operation set into J and $\bar{J}$. The processing times of the operations that are in set $\bar{J}$ are set to their lower bounds. What remains is to determine the optimal processing times of the remaining operations that are in set J. These can be determined using the following lemma. Let $\Gamma=\sum_{h\in\bar{J}}t_h^L$.

Lemma 4. *In the optimal solution of the $\varepsilon(F_1|\hat{T})^{S_1}$, the processing times of the operations in set J satisfy the following system of nonlinear equations*:

1. $t_l^*=\partial f_l^{-1}(\partial f_k(t_k)|_{t_k=t_k^*})$, $\forall l,k\in J$,
2. $\sum_{l\in J}t_l^*=\hat{T}-\Gamma$.

Proof. As a consequence of Lemma 3, for a given value of $\hat{T}$, we can determine which operations are in J and which are in $\bar{J}$ in the optimal solution. Additionally, Lemma 1 suggests that the cycle time bound constraint is satisfied as equality in the optimal solution. As a result, $\varepsilon(F_1|\hat{T})^{S_1}$ reduces to the following:

$$\min \quad \sum_{l\in J}f_l(t_l)$$

$$\text{s.t.} \quad \sum_{l\in J}t_l=\hat{T}-\Gamma, \tag{8}$$

$$t_l\geq t_l^L, \quad \forall l\in J. \tag{9}$$

Note that any feasible vector $\boldsymbol{t}^*$ is regular for $\hat{T}>\hat{T}^L$. Since the objective function and the constraints are convex, any point satisfying the KKT conditions is optimal. Hence, we have $\partial f_l(t_l)|_{t_l=t_l^*}=-\lambda$, $\forall l\in J$ and $t_l^*=\partial f_l^{-1}(-\lambda)$. Using these, the processing time of any operation $l\in J$ can be represented in terms of another operation $k\in J$ as follows:

$$t_l^*=\partial f_l^{-1}(\partial f_k(t_k)|_{t_k=t_k^*}). \tag{10}$$

Finally, the processing times can be found using Eqs. (8) and (10) jointly. □

Note that we can select any one of the processing times as the basis and we can represent all other processing times in terms of this basis using Eq. (10). Inserting these into Eq. (8), we have a nonlinear equation of a single variable. Since the cost function is convex, this nonlinear equation can be solved using some search methods such as the bisection algorithm, Newton's method or the golden search algorithm, within an error bound. This is valid for any convex cost function. Depending on the structure of the cost function, it may also be possible to find a closed form solution to Eq. (10). In such a situation all processing times can be presented explicitly and determined without any error as illustrated in Example 1. As a consequence of

Lemmas 3 and 4, we can solve $\varepsilon(F_1|\hat{T})^{S_1}$ easily. This solution corresponds to one of the nondominated solutions on the efficient frontier. Hence, in order to get the overall picture of the efficient frontier, we can construct an algorithm that determines a total of r nondominated solutions, uniformly spread over the entire efficient frontier, where the c th solution has the following cycle time value:

$$\hat{T}^L + (c-1)\frac{\hat{T}^U - \hat{T}^L}{r-1}. \tag{11}$$

The algorithm which is named as EFFFRONT-S1 is used for this purpose. The SIMAM subroutine is called to determine the optimal processing times of each of the r solutions. This subroutine will also be used while determining the efficient frontier of the S_2 cycle.

Algorithm. EFFFRONT-S1:

Input: r, $t_l^L, f_l(\cdot)$, $l \in O$.
Output: $t_{lc}^*, l \in O$ with corresponding cycle time $\hat{T}_c$ and cost C_c values for $c = 1, 2, \ldots, r$.
1. $c \leftarrow 1$.
2. Calculate t_l^U satisfying $\partial f_l(t_l)|_{t_l = t_l^U} = 0$, $l \in O$.
3. Determine M_l, $l \in O$ (use Eq. (7)),
4. $\hat{T}_c \leftarrow \hat{T}^L + (c-1)\dfrac{\hat{T}^U - \hat{T}^L}{r-1}$ (use Eqs. in (6)).
5. **SIMAM**$(\boldsymbol{O}, \hat{\boldsymbol{T}}_{\boldsymbol{c}})$. Let t_l^*, $l \in O$, be the output.
6. $t_{lc}^* = t_l^*, l \in O$ and $C_c = \sum_{l \in O} f_l(t_l^*)$. Output t_{lc}^*, $\hat{T}_c$ and C_c, $l \in O$.
7. $c \leftarrow c + 1$.
8. If $(c \leq r)$, go to Step 4. Else, STOP.

Subroutine. SIMAM:

Input: $O' \subseteq O$, T.
Output: t_l^*, $l \in O'$.
1. Determine M_l for $l \in O'$ (use Eq. (7)),
2. Construct sets J and $\bar{J}$ according to Lemma 3. Set $t_h^* = t_h^L$, $h \in \bar{J}$,
3. Calculate $\Gamma = \sum_{h \in \bar{J}} t_h^L$,
4. Solve $T - \Gamma = \sum_{l \in J} \partial f_l^{-1}(\partial f_{\hat{k}}(t_{\hat{k}}))$, as prescribed in Lemma 4 to determine $t_{\hat{k}}^*$ for an arbitrary $\hat{k} \in J$:
5. Determine $t_l^* = \partial f_l^{-1}(\partial f_{\hat{k}}(t_{\hat{k}})|_{t_{\hat{k}} = t_{\hat{k}}^*}), l \in J, l \neq \hat{k}$,
6. Output $t_l^*, l \in O'$.

The following example focuses on the CNC turning operations which possess strictly convex nonlinear cost functions.

Example 1. Let us consider a 2-machine robotic cell with CNC turning machines. The manufacturing cost for these operations can be written as follows: $f_l(t_l) = C_o t_l + K_l U_l t_l^{a_l}$. Here C_o is the operating cost of the CNC machine ($/minute), $K_l > 0$ and $a_l < 0$ are specific constants for the required cutting tool to perform operation l and $U_l > 0$ is a specific constant for operation l regarding parameters such as the length and the diameter of the operation. Hence, $C_o t_l$ is the operating cost of the CNC machine and $K_l U_l t_l^{a_l}$ is the tooling cost. The optimal processing time of an operation $\hat{k} \in J$ can be determined by solving the following nonlinear equation:

$$\hat{T} - \Gamma = \sum_{l \in J} t_{\hat{k}}^{(a_{\hat{k}}-1)/(a_l-1)} \left(\frac{K_{\hat{k}} U_{\hat{k}} a_{\hat{k}}}{K_l U_l a_l} \right)^{1/(a_l-1)}.$$

Then t_l^* can be determined using $t_{\hat{k}}^*$ by solving the following:

$$t_l^* = t_{\hat{k}}^{*((a_{\hat{k}}-1)/(a_l-1))} \left(\frac{K_{\hat{k}} U_{\hat{k}} a_{\hat{k}}}{K_l U_l a_l} \right)^{1/(a_l-1)}, \quad \forall l \in J.$$

If all of the operations use the same tool type, then $K_l = K_k = K$ and $a_k = a_l = a$, $\forall l, k$. As a consequence, the optimal processing times of the operations that are in set J can be determined using the following:

$$t_k^* = \frac{(\hat{T} - \Gamma)(U_k)^{1/(1-a)}}{\sum_{l \in J} U_l^{1/(1-a)}}, \quad \forall k \in J. \tag{12}$$

In order to further clarify the discussion, consider the following numerical example where the same type of tool is used for all operations. Assume $C_o = 0.5$, $K = 4$, and $a = -1.49$. Assume each of the identical parts requires five operations and the following parameter values are provided: $\{l|t_l^L, t_l^U, U_l\} = \{1|1.2, 4.7, 3.96\}$, $\{2|2, 2.8, 1.12\}$, $\{3|1.8, 5.6, 5.93\}$, $\{4|3.5, 4.2, 3.53\}$, $\{5|2.2, 3.4, 1.67\}$. Also assume $\varepsilon = 1$ and $\delta = 2$. According to these parameters, $T_{S_1}^L = 28.7$ and $T_{S_1}^U = 38.7$. Subtracting $6\varepsilon + 6\delta = 18$ from these two, we get $\hat{T}^L = 10.7$ and $\hat{T}^U = 20.7$, respectively. Let us determine the optimal processing times of the operations for $T_{S_1} = 32.5$ or $\hat{T} = 14.5$. Using Eq. (7), $M_1 = 10.7$, $M_2 = 15.08$, $M_3 = 11.03$, $M_4 = 16.27$, and $M_5 = 14.47$. Since $\hat{T} = 14.5 > M_1, M_3, M_5$, according to Lemma 3, $J = \{1, 3, 5\}$ and $\bar{J} = \{2, 4\}$. Hence, $t_2^* = t_2^L = 2$, $t_4^* = t_4^L = 3.5$, and $\Gamma = t_2^L + t_4^L = 5.5$. Using Eq. (12), the optimal processing times of the remaining operations are $t_1^* = 3.122$, $t_3^* = 3.67$, and $t_5^* = 2.207$. The corresponding total manufacturing cost is $\sum_{l=1}^5 (C_o t_l + K U_l t_l^a) = 19.4039$.

4. Heuristic procedure for the S_2 cycle

In this section we will consider the S_2 cycle for which, unlike in the previous section, we also have to deal with the allocation problem. The allocation of the operations to the two machines means partitioning set O into two nonempty subsets O_1, O_2 such that $O_1 \cup O_2 = O$ and $O_1 \cap O_2 = \emptyset$. O_i denotes the set of operations that are allocated to machine i, $i = 1, 2$. The total processing time of the part on machine i is $P_i = \sum_{l \in O_i} t_l$, $i = 1, 2$. Using the binary variable x_{li}, which indicates whether operation l is allocated to machine i or not, we can also write P_i as $\sum_{l=1}^p x_{li} t_l$. Akturk et al. [1] prove that the operation allocation problem for the S_2 cycle, even when the processing times are fixed and there is only a single criterion, is *NP*-complete. Hence, we will develop a heuristic procedure that approximates the efficient frontier and perform a computational study to verify its solution quality. We will compare the results of the heuristic procedure by solving the epsilon-constraint problem using commercial nonlinear mixed integer problem solvers GAMS-DICOPT2x-C and GAMS-BARON 7.2.3. Before proceeding with the heuristic procedure let us first consider the mathematical formulation of the problem. For the S_2 cycle the cycle time bound can be written as $\max\{6\varepsilon + 8\delta, 4\varepsilon + 4\delta + \sum_{l=1}^p x_{l1} t_l, 4\varepsilon + 4\delta + \sum_{l=1}^p x_{l2} t_l\} \leq T$, which can be replaced by the following constraints:

$$4\varepsilon + 4\delta + \sum_{l=1}^p x_{li} t_l \leq T, \quad i = 1, 2, \tag{13}$$

$$6\varepsilon + 8\delta \leq T. \tag{14}$$

Since both x_{li} and t_l are decision variables, the first constraint is nonlinear. Let N_l denote a sufficiently large number. By replacing $x_{li} t_l$ with w_{li}, we can properly linearize the above constraints as follows:

$$4\varepsilon + 4\delta + \sum_{l=1}^p w_{li} \leq T, \tag{15}$$

$$w_{li} \geq t_l - N_l(1 - x_{li}), \tag{16}$$

$$w_{li} \leq t_l + N_l(1 - x_{li}), \tag{17}$$

$$w_{li} \leq N_l x_{li}, \tag{18}$$

$$w_{li} \geq 0. \tag{19}$$

Note that N_l must be greater than t_l. Hence, we will use $N_l = t_l^U$. As a result, the epsilon-constraint problem for the S_2 cycle can be formulated as follows:

$$\varepsilon(F_1|T)^{S_2}: \min \quad \sum_{l=1}^{p} f_l(t_l)$$

$$\text{s.t.} \quad (14)\text{–}(19)$$

$$x_{l1} + x_{l2} = 1, \quad \forall l, \tag{20}$$

$$t_l \geq t_l^L, \quad \forall l, \tag{21}$$

$$x_{li} \in \{0,1\}, \quad \forall l, \ \forall i. \tag{22}$$

This formulation is a Mixed Integer Nonlinear Programming Model (MINLP) which allocates the operations to the machines and determines processing time values for all operations guaranteeing a given cycle time value while minimizing the total manufacturing cost. Let O_i^*, $i = 1,2$, denote the set of operations allocated to machine i in the optimal solution of $\varepsilon(F_1|T)^{S_2}$. The following lemma characterizes some properties of the optimal solution. Let $\boldsymbol{t}^* = (t_1^*, \ldots, t_p^*)$ denote the vector of optimal processing times to $\varepsilon(F_1|T)^{S_2}$ throughout this section.

Lemma 5. *One of the following holds*:

1. $\sum_{l\in O_1^*} t_l^* = \sum_{l\in O_2^*} t_l^*$ *or*,
2. $\sum_{l\in O_1^*} t_l^* < \sum_{l\in O_2^*} t_l^*$ *and* $t_l^* = t_l^U, \forall l \in O_1^*$ *or*,
3. $\sum_{l\in O_2^*} t_l^* < \sum_{l\in O_1^*} t_l^*$ *and* $t_l^* = t_l^U, \forall l \in O_2^*$.

Proof. Let $\boldsymbol{t}^*$ correspond to an objective function value F_1^*. Assume that this solution satisfies $\sum_{l\in O_1^*} t_l^* < \sum_{l\in O_2^*} t_l^*$ and $t_l^* < t_l^U$, for at least one operation $\hat{l} \in O_1^*$. Then we have another feasible solution such that $\hat{t}_l^* = t_l^*$, $\forall l \neq \hat{l}$ and $\hat{t}_{\hat{l}}^* = t_{\hat{l}}^* + \beta$ for some β such that $0 < \beta \leq \min\{\sum_{l\in O_2^*} t_l^* - \sum_{l\in O_1^*} t_l^*, t_{\hat{l}}^U - t_{\hat{l}}^*\}$. Since the cost function is decreasing, $\hat{F}_1^* < F_1^*$. This contradicts with $\boldsymbol{t}^*$ being an optimal solution. We conclude that we have either $\sum_{l\in O_1^*} t_l^* \geq \sum_{l\in O_2^*} t_l^*$ and $t_l^* < t_l^U$ for at least one operation $\hat{l} \in O_1^*$ or $\sum_{l\in O_1^*} t_l^* < \sum_{l\in O_2^*} t_l^*$ and $t_l^* = t_l^U, \forall l \in O_1^*$. A similar argument will lead to a contradiction in the case $\sum_{l\in O_1^*} t_l^* > \sum_{l\in O_2^*} t_l^*$ and $t_{\breve{l}}^* < t_{\breve{l}}^U$ for at least one operation $\breve{l} \in O_2^*$. In conclusion, either $\sum_{l\in O_2^*} t_l^* \geq \sum_{l\in O_1^*} t_l^*$ and $t_{\breve{l}}^* < t_{\breve{l}}^U$ for at least one operation $\breve{l} \in O_2^*$ or $\sum_{l\in O_2^*} t_l^* < \sum_{l\in O_1^*} t_l^*$ and $t_l^* = t_l^U, \forall l \in O_2^*$. □

This lemma states that in the optimal solution of $\varepsilon(F_1|T)^{S_2}$ the operations are allocated such that the total processing times on both machines are as close to each other as possible while respecting the upper bound limitations on processing times. As a consequence, if the processing times are not decision variables, but predetermined parameters, the optimal allocation of operations to the machines can be determined by solving the following Mixed Integer Programming Problem (MIP) formulation:

$$\text{Allocation Problem (AP)} \min \quad T$$

$$\text{s.t.} \quad 6\varepsilon + 8\delta \leq T, \tag{23}$$

$$4\varepsilon + 4\delta + \sum_{l=1}^{p} x_{li}\hat{t}_l \leq T, \quad i = 1,2, \tag{24}$$

$$x_{l1} + x_{l2} = 1, \quad \forall l, \tag{25}$$

$$x_{l1}, x_{l2} \in \{0,1\}, \quad \forall l. \tag{26}$$

In this formulation, $\hat{t}_l$ is a parameter. We can make use of this formulation in order to determine the upper and lower bounds of the cycle time of the S_2 cycle, $T_{S_2}^U$ and $T_{S_2}^L$, respectively. Let $\boldsymbol{t^L} = (t_1^L, t_2^L, \ldots, t_p^L)$ and $\boldsymbol{t^U} = (t_1^U, t_2^U, \ldots, t_p^U)$ be the vectors of lower and upper bounds of the processing times of operations, respectively. Let T_L^* denote the optimal objective function value of the AP where $\hat{t}_l = t_l^L$, $\forall l$. Then, $T_{S_2}^L = T_L^*$. Furthermore, if the processing times on both of the machines are equal to each other in the optimal solution, then the point $(F_1(\boldsymbol{t^L}), F_2(\boldsymbol{t^L}))$, where $F_2(\boldsymbol{t^L}) = T_L^*$ is one of the nondominated solutions of the bicriteria formulation for S_2. This is the minimum cycle time-maximum cost solution. On the other hand, if the processing times on both machines are not equal to each other, according to Lemma 5, the point $(F_1(\boldsymbol{t^L}), F_2(\boldsymbol{t^L}))$ is dominated. Hence, we can conclude that solving the above AP provides a lower bound for the cycle time. Additionally, if the processing times on both machines are equal to each other, we get the nondominated solution corresponding to the minimum cycle time value. However, if the processing times are not equal to each other, in order to get the nondominated solution, the epsilon-constraint problem is solved by setting the cycle time bound to $T_{S_2}^L$. Similarly the upper bound of the cycle time can be found by solving the AP when $\hat{t}_l = t_l^U$, $\forall l$. The optimal objective function value of this formulation is the upper bound of the cycle time, $F_2(\boldsymbol{t^U})$. Note that, according to cases (2) and (3) of Lemma 5, the point $(F_1(\boldsymbol{t^U}), F_2(\boldsymbol{t^U}))$ is a nondominated solution for the bicriteria problem, which corresponds to the maximum cycle-time minimum cost pair.

Let x_{li}^* be the optimal solution of the AP where all processing times are set to their upper bounds, $\hat{t}_l = t_l^U$, $\forall l \in O$.

Lemma 6. *If* $\sum_{l=1}^{p} x_{li}^* t_l^U \leq 2\varepsilon + 4\delta$, $i = 1,2$, *then the efficient frontier consists of the single point* $(F_1(\boldsymbol{t^U}), F_2(\boldsymbol{t^U}))$, *where* $F_1(\boldsymbol{t^U}) = \sum_{l=1}^{p} f_l(t_l^U)$ *and* $F_2(\boldsymbol{t^U}) = 6\varepsilon + 8\delta$.

Proof. If $\sum_{l=1}^{p} x_{li}^* t_l^U \leq 2\varepsilon + 4\delta$, $i = 1,2$, then constraint (23) becomes binding in the optimal solution and the cycle time is equal to its minimum possible value of $6\varepsilon + 8\delta$. Furthermore, $\sum_{l=1}^{p} f_l(t_l^U)$ is the lower bound for the cost. Since both objectives attain their absolute minimum values at this solution, the point is certainly nondominated. □

The cycle time of the S_2 cycle cannot be less than $6\varepsilon + 8\delta$. This is the time required for the robot to perform all the necessary loading/unloading and transportation operations. In this cycle, while the processing on one machine continues, the robot does not wait in front of this machine, but loads/unloads the other machine. Hence the processing time of the operations on this machine can be increased so that the processing of all operations are completed when the robot arrives to unload this machine. Such a change does not increase the cycle time but reduces the cost. However, if the upper bounds of the processing times of the operations are small such that the total processing on both machines are completed before the robot arrives in front of the machines, even if all processing times are set to their upper bounds, there is a nondominated solution, as stated in the lemma above.

In order to estimate the efficient frontier, we will determine r uniformly spread nondominated solutions. The quality of the estimation depends on the magnitude of r. Since the allocation problem is NP-complete, solving even only one problem to optimality using a nonlinear mixed integer solver will require a significant computational effort. To this end, we will present a heuristic algorithm. This algorithm generates a new nondominated solution using the solution at hand. Starting from $T_{S_2}^L$, the cycle time value is incremented at each step until we reach $T_{S_2}^U$. Instead of using a fixed increment amount, a new increment amount is determined at each step

Table 1
(a) The while loop of the DM; (b) the backtracking iterations.

(a) Step	Ordered set	$a_j - a_k$
1	{10, 8, 7, 4, 3}	$10-8=2$
2	{7, 4, 3, 2}	$7-4=3$
3	{3, 3, 2}	$3-3=0$
4	{2, 0}	$2-0=2$
5	{2}	STOP
(b) Step	Partition	
5	∅	{2}
4	{0}	{2}
3	{3}	{2, 3}
2	{3, 4}	{2, 7}
1	{3, 4, 8}	{7, 10}

depending on problem characteristics. As a result, the number of nondominated solutions to be generated is unknown. The points will be spread in the range between $(F_1(\boldsymbol{t}^{\boldsymbol{L}}), F_2(\boldsymbol{t}^{\boldsymbol{L}}))$ and $(F_1(\boldsymbol{t}^{\boldsymbol{U}}), F_2(\boldsymbol{t}^{\boldsymbol{U}}))$.

The problem has two phases: finding the allocation of the operations to the machines and determining the processing time values of the operations. According to Lemma 5, the former of these problems is equivalent to the set partitioning problem, for which we will make use of the Difference Method (DM) developed by Karmarkar and Karp [9] which outperforms other existing polynomial time approximation algorithms from an average behavior perspective [11]. Let $D=\{a_1, a_2, \ldots, a_p\}$ be a set of numbers to be partitioned and $D'=D\backslash\{a_j, a_k\}\cup\{|a_j-a_k|\}$. From a partition (A', B') of D' a partition (A, B) of D can be constructed easily so that both partitions have identical differences. Suppose $a_j > a_k$ and $|a_j - a_k| \in A'$. Then

$$A = A'\backslash\{|a_j - a_k|\}\cup\{a_j\}, \quad B = B'\cup\{a_k\} \tag{27}$$

gives the desired partition.

Algorithm. Difference Method (DM):

Input: A set of numbers to be partitioned: D.
Output: A partition (A, B) of set D.
  while $|D| > 1$ **do**
    pick the largest two numbers a_j, $a_k \in D$.
    $D \leftarrow D\backslash\{a_j, a_k\}\cup\{|a_j - a_k|\}$.
  end while
  Do the backtracking operations as explained in Eq. (27).

Example 2. Let us consider partitioning set $D = \{7, 4, 8, 10, 3\}$. The steps of the algorithm can be traced in Table 1a. The resulting set with only 2 as the element in Step 5 of the algorithm is found by differencing operation (2−0=2). Hence, one of the sets will have "2" and the other will have "0" as an element. In this way the partition in Step 4 ({0}, {2}) will have identical difference as Step 5 (∅, {2}). In Step 3, "0" is found by differencing operation $(3-3=0)$. Hence, we will replace "0" by "3" and place the other "3" to the other set so that the difference between the two sets is still the same. Our new partition is now ({3}, {2, 3}). In Step 2, "3" is found by differencing operation (7−4=3). Hence, we will replace "3" by "7" and place a "4" to the other set. There are two 3's. We may select any one arbitrarily. This suggests that there are alternative partitions. Our new partition is now ({3, 4}, {2, 7}). In Step 1, "2" is found by differencing operation (10−8=2). We will replace "2" by "10" and place "8" to the other set. Our new partition is now ({3, 4, 8}, {7, 10}). The backtracking steps can be seen in Table 1b.

As already mentioned, a different increment value is used at each step of the algorithm in order to generate a new nondominated solution. More specifically, let T_c be the cycle time value of the current nondominated solution. A new solution is determined with a cycle time value of $T_c + inc$, where inc is the amount of increment calculated for that step. We determine two candidates and select the minimum of these as the increment value. One of these candidates is calculated considering the DM algorithm. Recall that this algorithm works with a set of fixed numbers. However, in our case the processing times which are to be partitioned are decision variables. In such a case, we have to determine the sensitivity of the algorithm. That is, for each processing time t_l, let β_l be calculated in such a way that adjusting the processing time as $t_l + \lambda$ does not change the allocation resulting from the DM algorithm for $0 < \lambda \leq \beta_l$, but it may for $\lambda > \beta_l$. These breakpoints are determined considering the ordering of the numbers at each step of the algorithm. The following example illustrates a procedure for calculating these breakpoints.

Example 3. Let us consider the same example above with $a_1 = 10$, $a_2 = 8$, $a_3 = 7$, $a_4 = 4$ and $a_5 = 3$. In Step 1 we have the following descending order of numbers: (10–8–7–4–3). Increasing $a_1 = 10$ does not yield a change in the ordering of the operations. However, increasing a_2 more than $a_1 - a_2 = 10 - 8 = 2$ yields a new order and possibly a new partition. Thus, $\beta_{21} = 2$, where β_{lj} is the bound for a_l at step j. Similarly, if we increase $a_3 = 7$ by more than $8 - 7 = 1$ unit, $a_4 = 4$ by more than $7 - 4 = 3$ units, or $a_5 = 3$ by more than $4 - 3 = 1$ units we may end up with a different partition. Hence, at this step we have $\beta_{11} = \infty$, $\beta_{21} = 2$,$\beta_{31} = 1$, $\beta_{41} = 3$ and $\beta_{51} = 1$. In Step 2 of the DM algorithm, a_1 and a_2 are removed from the set but a new element $(a_1 - a_2)$ is included. The ordering at this step is (7–4–3–2) resulting in $\beta_{32} = \infty$, $\beta_{42} = 3$, $\beta_{52} = 1$. Note that the last element, namely 2, is found as $a_1 - a_2 = 10 - 8 = 2$. Increasing this new element by more than $3 - 2 = 1$ units may lead to a new partition. Since this element was not in the original set we can only change its value by changing the values of the elements used to calculate it. Increasing this element by 1 unit can only be done by increasing $a_1 = 10$ by 1 unit. Hence, 1 is a bound for $a_1 \Rightarrow \beta_{12} = 1$. On the other hand, decreasing this new element by 2 or equivalently increasing $a_2 = 8$ by 2 leads to a negative number. Hence, $\beta_{22} = 2$. Proceeding in a similar manner the following bounds can be determined at each step:

Step	Bounds				
1	$\beta_{11}=\infty$	$\beta_{21}=2$	$\beta_{31}=1$	$\beta_{41}=3$	$\beta_{51}=1$
2	$\beta_{12}=1$	$\beta_{22}=2$	$\beta_{32}=\infty$	$\beta_{42}=3$	$\beta_{52}=1$
3	$\beta_{13}=1$	$\beta_{23}=2$	$\beta_{33}=0$	$\beta_{43}=0$	$\beta_{53}=0$
4	$\beta_{14}=\infty$	$\beta_{24}=2$	$\beta_{34}=2$	$\beta_{44}=2$	$\beta_{54}=2$

In the example above we illustrated a procedure for calculating a bound for each element at each step of the DM algorithm. Using these bounds, we can calculate an overall bound for each element in the original set. Given an allocation that has resulted from the DM algorithm, if we increase the value of one of the elements within its respective bound, then the DM algorithm will yield the same allocation as before. The bound for element l is $\beta_l = \min_{j,\beta_{lj}\neq 0}\{\beta_{lj}\}$. Note that, in some steps, the bounds may be equal to 0. This happens if there are alternative partitions with identical differences. These are not considered while calculating the overall bounds for the numbers. Furthermore, the processing times to be partitioned have upper bounds. Let a_l^U be the upper bound of a_l. Additionally we must have $0 < \beta_l \leq a_l^U - a_l$. Finally, the candidate for the increment value is $\min_l\{\beta_l\}$. This candidate for the increment value determines whether we need to run the DM algorithm once more or not in order to generate the new solution.

The second candidate for the increment comes from Lemma 3. This lemma determines whether the processing time of an operation is equal to its lower bound or not by making use of the M_l values. It is important to note that the calculation of the M_l values for the S_2 cycle differs from the S_1 cycle since the cycle time of S_1 does not depend on

the allocation of the operations to the machines. Therefore, M_l values for the S_2 cycle should be calculated for a given allocation as follows: $M_l = \sum_{h\in O_i} \max[t_h^L, \partial f_h^{-1}(\partial f_l(t_l)|_{t_l=t_l^L})]$, $\forall l \in O_i$, $i=1,2$. From constraint (15) of $\varepsilon(F_1|T)^{S_2}$ the sum of the processing times allocated to the same machine must be less than or equal to $T_c-4\varepsilon-4\delta$. If $M_l > T_c-4\varepsilon-4\delta$ for an arbitrary operation l, then $t_l^* = t_l^L$. Assume that the cycle time is incremented so that $T_{new} = T_c + \lambda$. If $\lambda \le M_l - T_c - 4\varepsilon - 4\delta$, then $t_l^* = t_l^L$. Otherwise, $t_l^* > t_l^L$. As a consequence, the second candidate is calculated as $\Delta_T = T_c - 4\varepsilon - 4\delta - \max_{l\in\bar{J}}\{M_l\}$. Finally, the increment value for the cycle time to determine a new nondominated solution is determined as $\min\{\Delta_T, \min_l\{\beta_l\}\}$. We have observed that in some cases this increment value becomes very small especially when the number of operations is high. As a result, a set of unnecessarily large number of solutions is determined by the heuristic, resulting in relatively large CPU times. In order to avoid this, the increment value is bounded below by $\omega = E * 10^{\ln p/10}$. By this bound, the number of points generated by the algorithm and the CPU time requirements are balanced for problems with differing number of operations. The bound can be adjusted by altering the constant E.

Algorithm. A Heuristic Procedure to Construct the Efficient Frontier (EFFRONT-S_2)

Input: ω, t_l^L, $f_l(\cdot)$, $l \in O$.

Output: $t_{lc}^*, l \in O$ with corresponding cycle time T_c and cost C_c values and allocation of operations to machines (O_{1c}, O_{2c}), for each nondominated solution $c = 1, 2, \ldots, r$, and the total number of nondominated solutions r.

1. Set solution counter $c = 1$.
2. Calculate t_l^U satisfying $\partial f_l(t_l)|_{t_l=t_l^U} = 0$, $l \in O$.
3. **DM**$(\{\boldsymbol{t_1^U}, \boldsymbol{t_2^U}, \ldots, \boldsymbol{t_p^U}\})$. Let (O_1, O_2) be the output.
   3.1 If $\sum_{l\in O_i} t_l^U \le 2\varepsilon + 4\delta$, $i = 1,2$, then $T^U = 6\varepsilon + 8\delta$ and $C^L = \sum_{l\in O} f_l(t_l^U)$. Output t_l^U, T^U, C^L, (O_1, O_2), c. STOP.
   3.2 Else, $T^U = \max_i\{\sum_{l\in O_i} t_l^U + 4\varepsilon + 4\delta\}$, $C^L = \sum_{l=1}^p f_l(t_l^U)$.
4. **DM**$(\{\boldsymbol{t_1^L}, \boldsymbol{t_2^L}, \ldots, \boldsymbol{t_p^L}\})$. Let (O_1, O_2) be the output. Set $O_{ic} = O_i$, $i = 1,2$ and $t_{lc} = t_l^L, \forall l \in O$.
   4.1. If $P_{ic} = \sum_{l\in O_{ic}} t_{lc} < 2\varepsilon + 4\delta$, $i = 1,2$ then, go to Step 5.
   4.2. Else,
      4.2.1. Calculate M_l for $l \in O_1$ and $l \in O_2$ independently using Eq. (7).
      4.2.2. **Calculate-t** $((\boldsymbol{O_{1c}}, \boldsymbol{O_{2c}}), \boldsymbol{t_{lc}})$. Let t_l be the output.
      4.2.3. Set $t_{lc} = t_l, \forall l \in O$. Calculate $T_c = \max_i\{P_{ic} + 4\varepsilon + 4\delta\}$, $C_c = \sum_{l\in O} f_l(t_{lc})$. Output t_{lc}, T_c, C_c, (O_{1c}, O_{2c}). Let $c \leftarrow c + 1$.
5. Calculate M_l, for $l \in O_1$ and $l \in O_2$ independently using Eq. (7).
6. If $P_{ic} < 2\varepsilon + 4\delta$, $i = 1,2$, then set $\Delta_T = \min_i\{2\varepsilon + 4\delta - P_{ic}\}$, else $\Delta_T = \min_i\{P_{ic} - \max_{l\in\bar{J}}\{M_l\}\}, i = 1,2$.

Determine breakpoints β_l, $l \in O$ as explained in Example 3.

   6.1. If $\Delta_T < \min_l\{\beta_l\}$,
      6.1.1. $inc = \max\{\Delta_T, \omega\}$.
      6.1.2. Set $T_i = \min\{\sum_{l\in O_{ic}} t_{lc}, P_{ic} + inc\}, i = 1,2$.
      6.1.3. **SIMAM**$(\boldsymbol{O_{1c}}, \boldsymbol{T_1})$ and **SIMAM**$(\boldsymbol{O_{2c}}, \boldsymbol{T_2})$. Set $t_{lc} = t_l$, $\forall l \in O$.
   6.2. Else,
      6.2.1. $inc = \max\{\min_l\{\beta_l\}, \omega\}$.
      6.2.2. Let $l^* \in O_{i^*} = \text{argmin}_l\{\beta_l\}$. Set $T_{i^*} = P_{i^*} + inc$. **SIMAM**$(\boldsymbol{O_{i^*}}, \boldsymbol{T_{i^*}})$.
      6.2.3. **DM**$(\{\boldsymbol{t_{1c}}, \boldsymbol{t_{2c}}, \ldots \boldsymbol{t_{pc}}\})$. Let (O_1, O_2) be the output. Set $O_{ic} = O_i$, $i = 1,2$.
         6.2.3.1. If $P_{ic} < 2\varepsilon + 4\delta$, $i = 1,2$, then go to Step 5
         6.2.3.2. Else, **Calculate-t**$((\boldsymbol{O_{1c}}, \boldsymbol{O_{2c}}), \boldsymbol{t_{lc}})$. Let t_l be the output. Set $t_{lc} = t_l$, $\forall l \in O$.
7. Calculate $T_c = \max_i\{P_{ic} + 4\varepsilon + 4\delta\}$, $C_c = \sum_{l\in O} f_l(t_{lc})$. Output t_{lc}, T_c, C_c, (O_1, O_2).
8. If $T_c = T^U$, output c. STOP. Else, let $c \leftarrow c + 1$, go to Step 6.

Subroutine. Calculate-t:

Input: (O_1, O_2), t_l.
Output: t_l, $\forall l \in O$.

1. If $\sum_{l\in O_1} t_l = \sum_{l\in O_2} t_l$, output t_l, $\forall l \in O$, return.
2. Else if $\sum_{l\in O_1} t_l > \sum_{l\in O_2} \boldsymbol{t}_l$, then **SIMAM**$(\boldsymbol{O}_2, \sum_{l\in O_1} t_l)$, output t_l, $\forall l \in O$, return.
3. Else if $\sum_{l\in O_1} t_l < \sum_{l\in O_2} \boldsymbol{t}_l$, then **SIMAM**$(\boldsymbol{O}_1, \sum_{l\in O_2} t_l)$, output t_l, $\forall l \in O$, return.

The EFFRONT-S_2 algorithm generates a number of solutions on the efficient frontier of the S_2 cycle. A new nondominated solution is generated by making use of the current solution at hand. Starting from an initial solution, first an increment value for the cycle time is determined and then the corresponding allocation of the operations and the processing time values are determined. After making the initialization in Step 1, the upper bounds of the processing times of the operations are calculated in Step 2 of the algorithm. In Step 3, the largest cycle time-smallest cost solution is determined by setting all processing times to their upper bounds and allocating them to the machines. Similarly, in Step 4, the smallest cycle time-largest cost alternative is determined by setting all processing times to their lower bounds and allocating the operations to the machines. However, in this case, after determining the allocation, if the total processing time of one of the machines is less than the other one, the processing times of the operations allocated to this machine are updated using the SIMAM algorithm so that the total processing times on the machines become identical. In Step 5, the breakpoints that are necessary for Lemma 3 to construct sets J and $\bar{J}$ are calculated. In Step 6, the increment value for the cycle time in order to find the next efficient solution is determined and the new allocations and the new processing times of the operations are determined. More specifically, the difference between the current cycle time value and the next largest M_l value, $l \in \bar{J}$, is calculated as the first candidate for the increment value. The breakpoints which lead to a different allocation are calculated from the DM algorithm and the minimum of these is selected as the second candidate for the increment value. Finally, the minimum of these two candidates is selected as the increment value. If this increment is equal to the first candidate, then without changing the allocation of the operations, the new processing times of the operations are calculated. This is done by incrementing the total processing time of both machines with the selected increment value and applying the SIMAM algorithm to both machines independently. On the other hand, if the second candidate is selected as the increment value, then the machine, where the operation leading to this increment value is allocated, is determined. The total processing time of only this machine is updated by the selected increment value. Using the SIMAM algorithm, the new processing times of only the operations that are allocated to this machine are determined. Next, the operations are reallocated using the DM method. If in the resulting allocation one machine has a smaller total processing time value than the other one, the processing times of the operations allocated to this machine are updated using the SIMAM algorithm so that the total processing times on both machines become identical again. Note that, in any case, the increment value is bounded below by $E * 10^{\ln p/10}$. The heuristic continues until the cycle time value for the next solution to be determined is equal to the upper bound of the cycle time.

This heuristic procedure generates a set of nondominated solutions which are not necessarily equally spaced. However, a large number of points spread throughout the entire efficient frontier are generated and the distance between two consecutive points is very small. Additionally, any two solutions generated by the EFFRONT-S_2 cannot dominate each other. Indeed, the EFFRONT-S_2 algorithm generates a new solution starting with an initial solution. Let $T_c < T^U$ and C_c be the cycle time and cost values of a solution generated by the

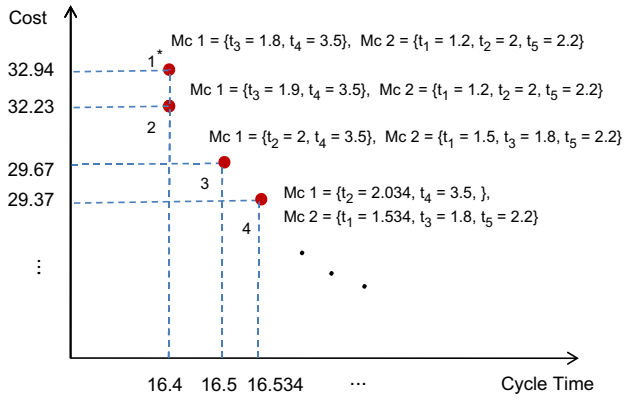

Fig. 1. Results of the first four iterations of the EFFRONT-S_2 algorithm for Example 4.

algorithm. Using this solution, a new solution is generated by setting $T_{(c+1)} = T_c + increment$, where $increment > 0$. Then, the EFFRONT-S_2 algorithm allocates this increment in cycle time to the individual processing times using the SIMAM algorithm as a subroutine, which guarantees the allocation to be the most cost reducing alternative (since the cost function is decreasing with respect to the processing times). As a result, $C_{(c+1)} < C_c$. This guarantees that $T_c < T_d$ and $C_c > C_d$ for any two solutions attained at iterations c and d, respectively, by this algorithm.

The following example illustrates some iterations of the algorithm and compares the S_1 cycle with that of S_2 for some data points.

Example 4. Let us consider the setting in Example 1 with $\{l|t_l^L, t_l^U, K_l * U_l, a_l\} = \{1|1.2, 4.7, 15.87, -1.49\}$, $\{2|2, 2.8, 4.48, -1.56\}$, $\{3|1.8, 5.6, 23.72, -1.46\}$, $\{4|3.5, 4.2, 14.13, -1.70\}$, $\{5|2.2, 3.4, 6.66, -1.38\}$ and $\varepsilon = 0.5$, $\delta = 1$. In Fig. 1, we depict the first four iterations of the EFFRONT-S_2 algorithm. In the first iteration (denoted as 1^*), the minimum cycle time-maximum cost solution is found by solving the AP formulation with commercial MIP solver GAMS-CPLEX 9.1 by setting all processing times to their lower bounds. Let T^* be the optimal objective function value of the AP. The solution found by CPLEX is a nondominated solution if the processing times on both machines are equal to each other. However, if the processing times are not equal to each other as in this example, in order to determine the minimum cycle time-maximum cost solution, the ECP formulation is solved by using T^* as the cycle time bound in iteration 2. This new solution dominates the previous one since the two solutions have identical cycle time values whereas the second solution has a smaller manufacturing cost than the first one. Note that both the allocation of the operations to the machines and the processing time values may change from one solution to another. Furthermore, as T increases, the processing times of some operations may decrease depending on the allocated machine. These properties reveal the complexity of the problem. In order to compare the S_1 and the S_2 cycles, let us use the same data and solve for both cycles. Using Eq. (6), one can calculate the minimum cycle time for the S_1 cycle by setting all processing times to their lower bounds. This solution has $T_{S_1} = 19.7$ and the corresponding cost is 32.94. For the same cycle time level, the S_2 cycle has a cost of 17.61. Since the S_2 cycle has a smaller cost than the S_1 cycle for the same cycle time, we say that the S_2 cycle dominates the S_1 cycle for this cycle time level. When we increase the cycle time to 21 time units and use the formulas derived in Example 1, the minimum cost of the S_1 cycle reduces to 25.33, whereas the minimum cost of the S_2 cycle is 17.22. Hence, the S_2 cycle dominates the S_1 cycle again. In fact, for the given data, the S_2 cycle dominates the S_1 cycle for all feasible cycle time values. However, if $\varepsilon = 1$ and $\delta = 6$ and all other data remain the same, the minimum cycle time that we can get with the S_1 cycle is 52.7 and the minimum cycle time for the S_2 cycle is 54 meaning that the $S1$ cycle dominates the $S2$ cycle for this cycle time level.

5. Computational study

In this section we will test the efficiency of the EFFRONT-S_2 algorithm under a specially tailored experimental design setting. This algorithm works for any strictly convex and differentiable function. In order to evaluate the algorithm we will consider CNC turning operations for which the cost function is presented earlier. In order to compare the results of the EFFRONT-S_2 algorithm we require a MINLP solver. One alternative is the GAMS-BARON 7.2.3. This commercial solver implements deterministic global optimization algorithms of the branch-and-bound type that are guaranteed to provide global optima under fairly mild assumptions. However, especially for large problem instances in terms of the number of operations, the CPU time requirement of BARON is huge and it is not possible to solve problems in a reasonable time. Hence, for large problem instances we need another alternative for generating "good quality" solutions in order to make a more comprehensive test of the performance of the EFFRONT-S_2 algorithm. For this purpose, commercial solver GAMS-DICOPT2X-c is selected. The MINLP algorithm inside DICOPT solves a series of NLP and MIP subproblems. Despite the speed of the algorithm, DICOPT is unable to prove the optimality and the quality of the solution provided. Hence, we also need to justify the solution quality of the DICOPT which will be done by comparing the results of DICOPT with BARON for small problem instances.

We coded the EFFRONT-S_2 algorithm in C language and compiled with Gnu C compiler. The DICOPT and EFFRONT-S_2 methods were run on a computer with 512 MB memory and Pentium IV 3.00 GHz CPU. However, due to licensing limitations, the BARON software is ran on a computer with 1294 MB memory and Pentium III 1133 MHz CPU.

As listed in Table 2, there are four experimental factors that can affect the efficiency of the methods. The number of operations affects the problem size and thus the computational requirements. The most important parameters that affect the efficiency of the methods are t^L and t^U. Using factors B, C and D, t^L and t^U parameters are generated using $t_l^U = U[C * D, D]$, and $t_l^L = B * t_l^U$, where $U[a, b]$ is a Uniform distribution on the interval $[a, b]$. Factors C and D affect the shape of the manufacturing cost function and this in turn affects the running times of the MINLP solvers. A small value for factor B means a greater range between t^L and t^U and a small value for factor C means greater variability for both t^L and t^U values in which case the MINLP solvers are expected to work better. The t^U level is another important parameter that increases the importance of the allocation of the operations to the machines. If the t^U level is high, then the penalty of a poor allocation is high in which case the performance of the DM algorithm used inside the EFFRONT-S_2 algorithm improves. In addition to these experimental design parameters, we assume identical CNC machines with operating cost $C_o = 0.5$. Consistent with earlier studies [10], a_l is selected from $U[-1.7, -1.3]$ and given these parameters, the required values for $K_l * U_l$ can be calculated using, $K_l * U_l = -C_o/a_l(t_l^U)^{(a_l - 1)}$, $\forall l \in \{1, \ldots, p\}$.

Also note that the robot transportation time δ and load/unload time ε do not have an effect on either the allocation of the operations or on the processing times. However, if these parameters are too large when compared to the processing times, this affects the values that the cycle time can attain. For example, when all processing times are set to their upper bounds and allocated to the machines, if the processing times on both machines are less than $2\varepsilon + 4\delta$, then the only value of the cycle time is $6\varepsilon + 8\delta$ as shown in Lemma 6. Hence, in order to test the efficiency of the EFFRONT-S_2 algorithm we assume

Table 2
Experimental design factors.

Factor	Definition	Level 1	Level 2	Level 3
A	Number of operations (p)	20	50	80
B	t^L–t^U range	0.5	0.8	
C	t^U variability	0.7	0.3	
D	t^U level	5	10	

Table 3
Completion statistics for DICOPT and BARON.

p	DICOPT			BARON		
	$optcr = 0$	$optcr = 0.05$	Time limit	$optcr = 0$	$optcr < 0.05$	$optcr = 0.05$
20	800	–	–	65 (CPLEX)	594	141
50	154	612	34	A	A	A
80	104	653	43	A	A	A

A: BARON could not find a solution within the relative optimality gap ($optcr = 0.05$) in 48 h.

Table 4
Summary of results.

p		R_1			R_2			R_3		
		>0	$=0$	<0	>0	$=0$	<0	>0	$=0$	<0
20	Number	90	6	704	387	2	411	699	101	0
	Min ($\times 10^{-6}$)	0.009	–	−0.006	0.007	–	−0.009	0.007	–	–
	Avg ($\times 10^{-6}$)	0.436	–	−31.402	0.785	–	−49.361	2.982	–	–
	Max ($\times 10^{-6}$)	6.721	–	−5722.0	8.337	–	−5722.0	67.585	–	–
50	Number	154	5	641						
	Min ($\times 10^{-6}$)	0.003	–	−0.003	A			A		
	Avg ($\times 10^{-6}$)	0.099	–	−2.435						
	Max ($\times 10^{-6}$)	0.341	–	−59.319						
80	Number	214	5	581						
	Min ($\times 10^{-6}$)	0.002	–	−0.002	A			A		
	Avg ($\times 10^{-6}$)	0.073	–	−26.095						
	Max ($\times 10^{-6}$)	0.245	–	−10 653.6						

A: BARON could not find a solution within the relative optimality gap ($optcr = 0.05$) in 48 h.

that $\varepsilon = 0$ and $\delta = 0$. In this way, we guarantee that the heuristic will not stop in STEP 2 and a comprehensive test can be made.

Five replications are taken for each of the experimental settings, $(3 * 2 * 2 * 2)$, which makes a total of 120 different problem settings, so that we will approximate a total of 120 efficient frontiers. In order to approximate these, we will use 20 different cycle time bounds that are spread over the entire efficient frontier, totaling to 2400 problems. The minimum cycle time-maximum cost and the maximum cycle time-minimum cost solutions are found by solving the AP formulation with commercial MIP solver GAMS-CPLEX 9.1 by setting all processing times to their lower bounds and to their upper bounds, respectively. The remaining 18 problems are solved using a MINLP solver. For small instances where $p = 20$ the model is solved using BARON. However, due to CPU restrictions, even for these small instances, it is not possible to run the models till the end. As a consequence, we run the BARON model with a time limit of 1800 s. If the run is not completed within this time limit but the relative optimality gap $optcr \leq 0.05$, the model is stopped immediately. Otherwise, it is run until $optcr = 0.05$. That is, when $optcr = 0.05$, the MINLP model stops as soon as $(Best\ Found - Best\ Possible)/Best\ Possible \leq 0.05$.

Similarly, for the DICOPT model which will be used to test our algorithm for large instances, we first run the model with $optcr = 0$ for 900 s. If the model is not completed within this time limit, we set $optcr = 0.05$ and continue the run for an additional 3600 s. As a consequence, the solver either makes a normal completion before 900 s or stops with $optcr = 0.05$ between somewhere in [900, 4500] s after starting the run or stops due to time limit when it reaches to 4500 s.

Table 3 lists the number of instances with each stopping criteria for both MINLP methods. Note that all of the completions where $optcr = 0$ listed for BARON are actually achieved by the CPLEX solver for the AP formulations to determine the minimum cost-maximum cycle time and maximum cost-minimum cycle time solutions. For the instances where $p = 50$ and 80, BARON was unable to find a feasible solution even with $optcr = 0.05$ in 48 h. For $p = 20$, none of the BARON runs are completed within the time limit when $optcr = 0$. Under DICOPT, for $p = 20$ all 800 instances including the CPLEX runs are completed within the 900 s time limit, whereas this number reduces to 154 and 104 for $p = 50$ and 80, respectively.

The experimental design parameters as well as the manufacturing cost parameters and the step size constant E jointly affect the number of approximate efficient points generated by the EFFRONT-S_2 algorithm. Since this number is not known in advance, in order to compare the results of different methods, we first run the EFFRONT-S_2 algorithm and generate a set of points. Then we choose 18 of these other than the minimum cycle time-maximum cost and maximum cycle time-minimum cost solutions such that each successive point pair has (almost) equal separation and run the MINLP models for the corresponding cycle time values of the 20 points. We use $E = 0.0001$. For notational simplicity let us denote $F_1(S, \boldsymbol{P})$ and $F_2(S, \boldsymbol{P})$ by F_1 and F_2, respectively. We measured the relative difference between F_1 values of different methods corresponding to identical F_2 values. Let

$$R_1 = (F_1(EFFRONT - S_2) - F_1(DICOPT))/F_1(DICOPT),$$

$$R_2 = (F_1(EFFRONT - S_2) - F_1(BARON))/F_1(BARON),$$

$$R_3 = (F_1(DICOPT) - F_1(BARON))/F_1(BARON).$$

Table 4 provides results that can be used to compare the methods in terms of their average, minimum and maximum R_1, R_2 and R_3 values. When we compare the EFFRONT-S_2 algorithm with BARON for the case when $p = 20$, we see that these two methods perform similar to each other. The number of instances where each of these methods performed better than the other is comparable (387 vs 411). However, the average value of R_2 is greater (in absolute magnitude) for the instances where EFFRONT-S_2 performed better than the instances where BARON performed better (−49.361 vs 0.785 ($\times 10^{-6}$), respectively). We cannot compare these two methods for $p = 50$ and 80 cases since we could not get a feasible solution using BARON even with $optcr = 0.05$ in 48 h. Hence, we will compare our algorithm with DICOPT for these problem instances. However, since DICOPT provides no information on the optimality gap, let us first compare DICOPT with BARON and evaluate its solution quality. In 699 out of 800 cases, BARON performed better than DICOPT and in the remaining ones they found the same solution. DICOPT could not find a single better solution than BARON. However, the average R_3 value is very small (2.982×10^{-6}). This small difference suggests that DICOPT can be used to evaluate the performance of the EFFRONT-S_2 algorithm for $p = 50$ and 80 cases. The R_1 statistics show that, although the average difference is very small (0.203×10^{-6}), in most of the cases the EFFRONT-S_2 algorithm finds better solutions than DICOPT. In 704, 641 and 581 out of 800 instances for 20, 50 and 80 operations, respectively, the EFFRONT-S_2 algorithm found a better solution than DICOPT. As the number of operations increases, this performance seems to slightly decrease. This is due to the use of the step size limit. This limit is 0.000069 for 20 operations case whereas it is 0.012 for 80 operations case with $E = 0.0001$. Using a smaller step size limit increases the number and present the quality of the generated points. On the other hand, a smaller step size means greater CPU time requirements.

Table 5
Number of points generated by the EFFRONT-S_2 and the CPU times.

p	Factors			N	EFFRONT-S_2		DICOPT	BARON
	B	C	D		CPU time (s)	Number of points	CPU time (s)	CPU time (s)
20	0	0	0	100	27.1	29 207.6	52.3	1983.1
	0	0	1	100	42.7	44 938.8	45.3	2006.2
	0	1	0	100	16.5	18 416.2	50.7	1577.3
	0	1	1	100	20.8	21 975.0	69.8	1636.3
	1	0	0	100	9.0	9716.8	129.2	1692.2
	1	0	1	100	20.1	19 871.2	121.4	1692.3
	1	1	0	100	4.8	4745.6	118.5	1696.0
	1	1	1	100	7.7	7324.0	93.6	1684.2
Average					18.6	19 524.4	85.1	1746.0
50	0	0	0	100	96.0	25 351.4	838.3	
	0	0	1	100	188.9	50 500.0	885.1	
	0	1	0	100	73.0	19 884.4	852.6	
	0	1	1	100	135.8	36 095.0	1060.2	
	1	0	0	100	38.1	9896.8	900.3	
	1	0	1	100	73.7	19 727.2	846.3	
	1	1	0	100	28.0	7763.4	873.2	
	1	1	1	100	53.4	14 440.4	882.1	
Average					85.9	22 957.3	892.3	
80	0	0	0	100	112.0	14 020.0	790.5	
	0	0	1	100	225.3	27 860.4	786.1	
	0	1	0	100	86.5	10 930.0	737.0	
	0	1	1	100	175.3	21 791.0	1857.6	
	1	0	0	100	43.4	5594.6	882.3	
	1	0	1	100	88.7	11 072.6	880.2	
	1	1	0	100	32.5	4240.6	882.4	
	1	1	1	100	66.2	8415.0	891.3	
Average					103.8	12 990.5	963.4	

Another factor for evaluating the quality of an algorithm is the CPU time requirements. The total number of points generated by the EFFRONT-S_2 algorithm and the corresponding CPU times for each factor combination for all methods are presented in Table 5. The CPU times listed for DICOPT and BARON are only for generating 20 points on the efficient frontier. The results indicate that in a very small CPU time, the EFFRONT-S_2 algorithm generates at least 600 times more points than what DICOPT and BARON could generate. The increase in the CPU time as the number of operations increases is very small for the EFFRONT-S_2 algorithm (from 18.6 for 20 operations to 203.7 for 80 operations). On the other hand, for DICOPT even for 50 operations, the average CPU time requirements (892 s) reaches time limit of normal completion (900 s) and for BARON we are not able to generate solutions for $p = 50$ and 80 cases even with setting $optcr = 0.05$ after 48 h.

6. Conclusion

In this study, we considered a bicriteria robotic cell scheduling problem in a 2-machine robotic cell with highly flexible CNC machines. Instead of assuming the processing times to be fixed on each machine, we assumed the allocations of the operations as well as their processing times to be decision variables. We formulated the problem as a Mixed Integer Nonlinear Programming (MINLP) model. We developed an exact solution procedure for the S_1 cycle. Since the allocation problem for the S_2 cycle is NP-Complete, we presented a heuristic algorithm that generates a set of approximate efficient solutions. We compared the results of the algorithm with commercial MINLP solvers DICOPT and BARON. The proposed algorithm proved to be very efficient in terms of the number and the quality of the generated solutions and the computational requirements.

References

[1] Akturk MS, Gultekin H, Karasan OE. Robotic cell scheduling with operational flexibility. Discrete Applied Mathematics 2005;145(3):334–48.
[2] Bretthauer KM, Shetty B. The nonlinear knapsack problem—algorithms and applications. European Journal of Operational Research 2002;138:459–72.
[3] Bertsekas DP. Nonlinear programming. Belmont, MA: Athena Scientific; 1999.
[4] Crama Y, Van de Klundert J. Cyclic scheduling of identical parts in a robotic cell. Operations Research 1997;45(6):952–65.
[5] Crama Y, Kats V, Van de Klundert J, Levner E. Cyclic scheduling in robotic flowshops. Annals of Operations Research 2000;96:97–124.
[6] Dawande M, Geismar HN, Sethi S, Sriskandarajah C. Sequencing and scheduling in robotic cells: recent developments. Journal of Scheduling 2005;8(5):387–426.
[7] Gultekin H, Akturk MS, Karasan OE. Bicriteria robotic cell scheduling. Journal of Scheduling 2008;11(6):457–73.
[8] Hoogeveen H. Multicriteria scheduling. European Journal of Operational Research 2005;167:592–623.
[9] Karmarkar N, Karp RM. The differencing method of set partitioning. Technical Report UCB/CSD 82/113, Computer Science Division, University of California, Berkeley, CA; 1982.
[10] Kayan RK, Akturk MS. A new bounding mechanism for the CNC machine scheduling problems with controllable processing times. European Journal of Operational Research 2005;167:624–43.
[11] Michiels W, Korst J, Aarts E, Leeuwen JV. Performance ratios for the Karmarkar–Karp differencing method. Journal of Combinatorial Optimization 2007;13(1):19–32.
[12] Sethi SP, Sriskandarajah C, Sorger G, Blazewicz J, Kubiak W. Sequencing of parts and robot moves in a robotic cell. International Journal of Flexible Manufacturing Systems 1992;4:331–58.
[13] Shabtay D, Steiner G. A survey of scheduling with controllable processing times. Discrete Applied Mathematics 2007;155(13):1643–66.